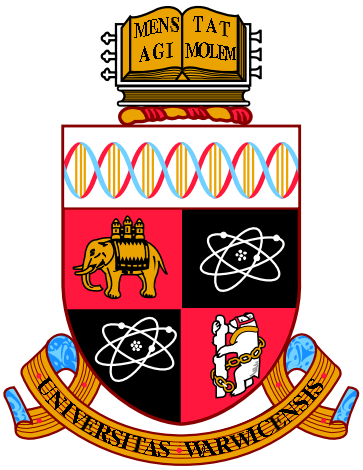


Low Computational SLAM for an Autonomous Indoor Aerial Inspection Vehicle

by

Stefan Winkvist

Thesis

Submitted to the University of Warwick

for the degree of

Doctor of Philosophy

WMG

November 2013

THE UNIVERSITY OF WARWICK

Acknowledgments

This project was funded by both *Sellafield Ltd* and the *Engineering and Physical Sciences Research Council* (EPSRC) through the former *Warwick Innovative Manufacturing Research Centre* (WIMRC).

I want to thank:-

My supervisors *Emma Rushforth* and *Ken Young* for their excellent help and guidance.

Sellafield Ltd, in particular *James Moore*, *Stephen Hepworth*, *Paul Mort* and *Samuel Abraham* including *Nick Mallinson* from the WIMRC for doing their utmost to aid and support me throughout the project.

The *University of Warwick*, the *WMG* and the *Manufacturing Technology Centre* (*MTC*) for allowing me to fly the UAV inside their buildings.

The undergraduate students of the *Warwick Mobile Robotics: Search & Rescue* project 2010 – 2013 whom I've spent many enjoyable hours lending a helping hand while repetitively subjecting them to the same bad jokes and terrible puns. In particular *James Williams* of the 2008 – 2009 team whom, during a demonstration of the WMR robot at an exhibition hosted by Sellafield, asked if they were interested in sponsoring a robotics Ph.D, effectively founding this project.

All my friends at Warwick who have all helped in some way, from offering a endless stream of procrastination to picking up pieces of UAV that one time it "decided" to hit a wall. In particular *Mark Hollands* and *Simon Walker* for volunteering to be operators during flight testing.

Lastly, a **special** thank-you to my parents for supporting me throughout, for enduring much whingeing and complaining during the writing of this thesis, and for their help with proof-reading.

Declarations

I hereby declare that:-

- This thesis and work is my own and that no part of this thesis has been published or submitted for publication elsewhere.
- Where other sources have been used, they have been acknowledged.

Abstract

The past decade has seen an increase in the capability of small scale *Unmanned Aerial Vehicle* (UAV) systems, made possible through technological advancements in battery, computing and sensor miniaturisation technology. This has opened a new and rapidly growing branch of robotic research and has sparked the imagination of industry leading to new UAV based services, from the inspection of power-lines to remote police surveillance.

Miniaturisation of UAVs have also made them small enough to be practically flown indoors. For example, the inspection of elevated areas in hazardous or damaged structures where the use of conventional ground-based robots are unsuitable. Sellafield Ltd, a nuclear reprocessing facility in the U.K. has many buildings that require frequent safety inspections. UAV inspections eliminate the current risk to personnel of radiation exposure and other hazards in tall structures where scaffolding or hoists are required.

This project focused on the development of a UAV for the *novel* application of semi-autonomously navigating and inspecting these structures without the need for personnel to enter the building. Development exposed a significant gap in knowledge concerning indoor localisation, specifically *Simultaneous Localisation and Mapping* (SLAM) for use on-board UAVs. To lower the on-board processing requirements of SLAM, other UAV research groups have employed techniques such as off-board processing, reduced dimensionality or prior knowledge of the structure, techniques not suitable to this application given the unknown nature of the structures and the risk of radio-shadows.

In this thesis a *novel* localisation algorithm, which enables real-time and three-dimensional SLAM running solely on-board a computationally constrained UAV in heavily cluttered and unknown environments is proposed. The algorithm, based on the *Iterative Closest Point* (ICP) method utilising approximate nearest neighbour searches and point-cloud decimation to reduce the processing requirements has successfully been tested in environments similar to that specified by Sellafield Ltd.

Abbreviations

ANN Approximate Nearest Neighbour

AUVSI Association for Unmanned Vehicle Systems International

CCD Charged Coupled Device

CoG Centre of Gravity

COTS Commercial Off-The-Shelf

DME Distance Measurement Equipment

FLANN Fast Library for Approximate Nearest Neighbour

FW Fixed Wing (Aircraft)

GPS Global Positioning System

GUI Graphical User Interface

IARC International Aerial Robotics Competition

ICP Iterative Closest Point

IMU Inertial Measurement Units

INS Inertial Navigation System

IR Infrared

LED Light Emitting Diode

LiDAR Light Detection And Ranging (Laser Rangefinders)

MEMS Micro-Electromechanical Systems

NDB Non-Directional Radio Beacon

NN Nearest Neighbour

PID Proportional Integral Derivative

PWM Pulse Width Modulation

RC Remote Control

RMSE Root Mean Square Error

SLAM Simultaneous Localisation and Mapping

SoNAR Sound Navigation And Ranging (Ultrasonic Rangefinders)

SVD Singular Value Decomposition

UAV Unmanned Aerial Vehicle

UGV Unmanned Ground Vehicles

UV Ultra-Violet

VOR Very High Frequency Omni-directional Radio Beacon

VTOL Vertical Take-Off and Landing

WMR Warwick Mobile Robotics

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the great challenges for nuclear facilities, be it for power generation, refining or storage, is that at some point in time they need to be decommissioned and dismantled. Modern facilities have been designed with this in mind, making the process of dismantling both quicker and cost-effective in the long run. However, early generations of plants built in the 1960's or earlier generally didn't follow this practice, allowing plants to be designed, built and operational in a fraction of the time and cost. Only now is this becoming a problem, as the older plants are nearing the end of their service life and require decommissioning.

An example of which is the legacy plant at Sellafield located in the Lake District, Cumbria (see Figure 1.1). Sellafield is the largest nuclear site in the United Kingdom, built initially to produce plutonium for the UK military weapons programme in 1950s [1]. Sellafield has also housed a number of prototype reactors, one of which in 1957 accidentally overheated, caught fire and subsequently released large amounts of radioactive material into the atmosphere[2].

In the late 1990s a newer reprocessing facility was built at the site, allowing decommissioning work to start on the older, disused legacy plant. Due to the complexity of the legacy plant and the way in which it was constructed it is estimated to take several decades to complete the decommission.

Figure 1.1: The Sellafield Site in 2009[3]

1.1 Nuclear Decommissioning

The process of decommissioning a nuclear facility includes[4]:-

1. Characterisation: Gathering data about the building being demolished.
2. Decontamination: Removal of hazardous waste.
3. Demolition: Progressively demolishing the structure
4. Waste Storage: Separating high, medium and low level radioactive waste for storage and allowing radioactivity to drop before “safe” disposal.

Firstly data needs to be gathered about the structural state of the building and its surroundings, along with a detailed analysis of the location, quantity and condition of any hazardous materials held within. Although classification is the first step of the

process, it is also a continuous task throughout the decommissioning project as over time the structure may decay or weaken, which for example may potentially lead to hazardous materials leaking from containers. It is therefore imperative that this information is kept up to date for the whole site, as situations may unexpectedly arise where a building may require immediate remedial work, resulting in other decommissioning projects to be temporarily put on hold.

Once data has been collected about the structure and its contents, a decontamination plan can be created. Depending on the level of radiation this may be done manually by sending workers into the building in protective clothing, or where levels are unsafe investigate other alternatives. However, this generally requires specialist one-off robotic or mechatronic solutions. One such example was required for decontaminating the Caesium extraction plant, where a robotic solution was developed (Figure 1.2) to decontaminate and dismantle the high-level radiation areas, allowing safe access for personnel to enter and complete the demolition process [5].

Even though remote decommissioning is preferred, generally lowering the risks and dose levels for the workers, using and developing these types of machines is a slow and expensive process and usually not portable or transferable to other sites or buildings.

The same problems also arise for classification, where it is primarily done by hand, relying on the workforce to either enter the buildings in protective gear, or through the use of cameras entered through drilled inspection holes. This method, although suitable for smaller, easily accessible structures poses a challenging problem for the larger structures, such as storage areas or chimneys, where the workforce has to rely on gantries or walk ways to safely access elevated areas of the structures. However, some of these structures have either not been fitted with the gantries when built, or the integrity of the gantries over the years has become questionable. In these cases additional hoists or scaffolding need to be installed. This is generally avoided as it too will become contaminated and require to be discarded as low level radioactive waste, further adding to the cost and complexity of the decommissioning effort. Not

Figure 1.2: Caesium Plant Dismantling Machine[5]

to mention the ever present risks of working at height, highlighted by the unfortunate incident where a worker was killed after falling from the upper section of one of the pile chimneys[6].

To address the issues, of helping lowering the dose of exposure to the workers and expediting the classification and decommissioning process, Sellafield Ltd has recently been looking for ways to remotely inspect the more difficult to reach areas. One way this has been attempted is through inviting businesses and universities to exhibitions to help showcase present technology and to get fresh ideas for possible solutions.

Back in 2008, the Warwick Mobile Robotics (WMR) group were invited to attend one of these exhibitions, both to demonstrate their search and rescue robot (Figure 1.3) and also to try to understand the needs and requirements for robotics in the nuclear industry, more specifically, nuclear decommissioning. With the intention, of a possible new direction for the WMR project in the years to come. What became apparent was that there were already many competing commercial robotic platforms available, many of which with developed radiation hardened bespoke sensor and

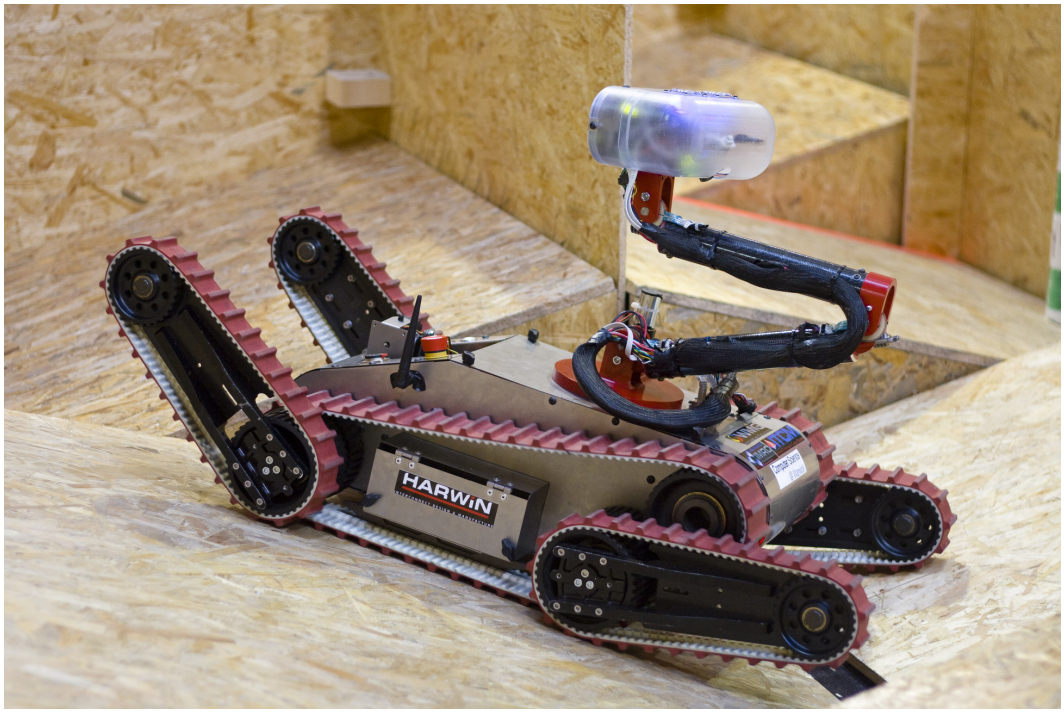


Figure 1.3: The Warwick Mobile Robotics: Search & Rescue Robot

actuator solutions.

A number of similar robots were later used in the Fukushima incident in Japan in 2011, although many failed to complete their mission of inspecting the reactor buildings. Common problems encountered were attributed to, for example lack of trained personnel to operate the vehicle due to the infrequent nature of these types of incidents, and umbilical power and data cables either breaking or snagging affecting the function of the robots requiring these [7].

The most successful robots were the military developed PackBOT and TALON, which entered the reactor building and successfully returned images and data of its interiors. Both robots however, were limited to observing from the ground floor, as there was hesitation allowing them to ascend or descend stairs [7].

During a discussion with the director of decommissioning at Sellafield it was pointed out, that despite all the commercial robots available there is a significant limitation to their use. They are all ground based and only able to inspect items near the

bottom of a structure, with no suitable solutions available to remotely inspect the internal details of taller structures. It is for this reason that this project was commissioned: to research and develop a *novel* robotic system which is capable of remotely inspecting these elevated areas.

1.2 Structure of the Thesis

As this project follows the conception and development of a complex system, the structure of this thesis is laid out to follow a systems V-Diagram approach to ensure a clear and easy to follow thought process behind all the vital system components shown in figure 1.4. The following chapter (Chapter 2) discusses the capability and requirements as given by Sellafield, which forms the foundation of the decision making behind the project and specifies the end goal.

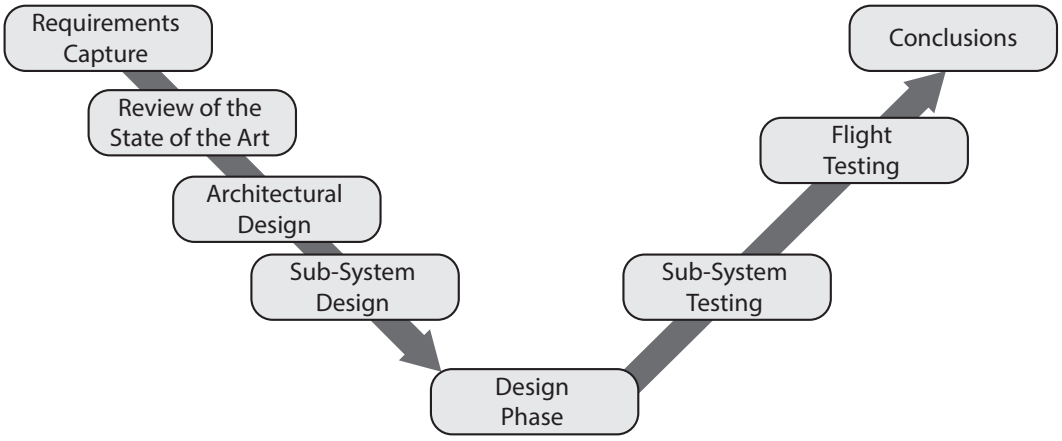


Figure 1.4: Systems V-Diagram

Once the requirements are understood it is important to review the current state-of-the-art. This is primarily done through a literature review to check if the requested system is achievable with current technology and whether a solution has already been developed, potentially for a different application. This forms Chapter 3 *"Review of the State of the Art"*. Chapter 4 *"Architectural Design"* takes the knowledge acquired from the literature review and specifies how the system, at a high level, should be

constructed. It also highlights the areas where the current state-of-the-art is lacking and needs to be developed further.

Chapter 5 *"Sub-System Design"* in detail explains how each of the chosen and developed systems function and any modifications made. Later moving back up the V-Diagram with verification and testing, firstly through *Sub-System Testing* in Chapter 6 where the key individual systems are tested. Later through whole system testing, in this case *Flight Testing* in Chapter 7 where the system is tested against the listed requirements.

Chapter 2

Requirements Capture

One of the most important steps in designing a system is knowing specifically what it has to be able to do, so that it can be both designed and tested against the same requirements. This chapter lists and discusses the primary aims and objectives of the project, of which are based on the requirements stated by Sellafield ltd at the start of the project.

2.1 Operating Environment

Sellafield's intended operating environment for the robot would be either medium sized buildings (where the longest distance between opposite walls is less than approximately 20-30m) with generally open interiors or a large chimney approximately 125 metres tall and 15 metre diameter[5] shown in figure 2.1. The structures may contain scaffolding and hanging cables which need to be avoided. There may also be considerable clutter both on the ground and on the walls, which although static during a mission may be moved between inspection flights, meaning the robot cannot accurately rely on using the same features for navigation between missions.

Another issue was that many buildings at Sellafield were not designed for easy worker access, occasionally resorting to seal-and-forget maintenance solutions during construction. This meant that, in order to get personnel or equipment into these building holes would need to be cut in the walls, so the smaller the designed robot could be, the better. Also, these structures are generally unlit or rely on temporary

Figure 2.1: Example of the internals of one of the Sellafield chimneys [5]

lighting solutions, so the robot should ideally not have to rely on external lighting to perform its task.

2.2 Inspection

With regards to inspection, Sellafield has requested that the robot should be able to record imagery, either through still photographs or video. A live feed was not required to be high-resolution as the imagery would be inspected post-flight, but was used to keep an eye on what the robot was doing and to align the robot with a point of interest.

The robot should also be equipped with a dosimeter, so that it could collect information about radiation levels, and possibly have a modular solution so that it could be swapped with other sensors as required, e.g. infrared camera, probes etc.

Lastly Sellafield were interested in obtaining geometric measurements (three-dimensional maps) concerning the structure and its contents, using this information they can

more accurately pinpoint the location of points of interest, or compare with older data to see if there has been any movement of items or parts of the structure between inspections.

2.3 Robotic Platform

The robot needed to be:-

1. As compact as possible to fit through inspection holes (approximately 30cm).
2. Have an operating time sufficient to monitor a couple of points of interest between charges.
3. Airborne and non-contact with any part of the structure.
4. Able to carry the required sensors and devices.
5. Cost less than £10,000 unless reliability and re-use can be guaranteed.

2.4 Autonomy

One of the primary criteria issued by Sellafield was that the robot should be easy to operate and maintain, requiring only minimal training in order to be used safely. This would increase the deployability of the system as there was a higher likely hood that someone is on site to operate it at short notice.

The robot however, needed to remain primarily under operator control as *full* autonomy was discouraged, the reasoning for this is two-fold. Firstly, programming and instructing the robot exactly what to look for would be more challenging than to let the operator tell the robot where to travel. Secondly, due to the on-site health and safety requirements imposed on these types of facilities, if used, the robot would need to be robust and reliable and all failure modes considered. A task

made more difficult for a fully autonomous robot where its “thought process” and response cannot necessarily be guaranteed, particularly in an unknown, untested environment.

Chapter 3

Review of the State of the Art

A review of the state of the art is a key step in developing a successful system. In-depth knowledge about the current state of technology and related projects not only helps highlight whether or not the system can feasibly be built (within the allotted time) but also results in a more capable system through intelligent design. Intelligent design being, for-instance, understanding the function of a chosen sensor, knowing its fundamental weaknesses/strengths and using this knowledge to better the synergy between itself, other sensors and the robotic platform.

A robot can generally be split into four separate yet heavily dependent sub-systems:-

1. The Robotic platform to which the sensors and other components are mounted.
2. Sensors - Used to give the robot sufficient exteroceptive and interoceptive data to provide sufficient knowledge of its surroundings and present state respectively.
3. Data processing and Autonomy - Analysing the sensor data to allow the robot to complete its task safely and in a timely manner.
4. Communication - If needed, sending and/or receiving data back to an operator or other robots.

This chapter aims to highlight and discuss some of the various options available for the sub-systems above. Later discussing their implementation through analysing similar projects in 3.5.

3.1 Flying Platforms

In the following section the different forms of aerial vehicles will be investigated, highlighting their advantages and disadvantages along with progress and use within current *Unmanned Aerial Vehicles* (UAVs) research. All aerial vehicles can be generalised into three distinct groups based on their method of creating lift, these are:-

- Lighter-than-air (e.g. blimps)
- Fixed wing (e.g. aeroplanes)
- Rotary wing (e.g. helicopters)

3.1.1 Lighter-than-Air

Lighter-than-air craft, as the name implies, specifies that the craft creates its lift through being positively or neutrally buoyant with the surrounding atmosphere. Historically this has been achieved through hot-air balloons, functioning on the principal that as air warms it becomes less dense and therefore rises above the surrounding air until it cools. Flight is achieved through holding a sufficiently large envelope of warm air, so that the buoyancy of the displaced air-mass is capable of lifting the payload.

Another method is to use a gas inherently lighter than air such as hydrogen or helium, although hydrogen is rarely used due to the reactive nature of the gas. This is the primary method of lift used for weather balloons and airships and offers a much higher ratio of lift/envelope volume compared to the hot-air method. Limited propulsion can be achieved through mounting motors, in the case of airships on the envelope, allowing the craft to be able to manoeuvre as opposed to just be carried by winds[8].

(a) Blended Wing[9]

(b) Biplane

(c) Glider

Figure 3.1: Examples of the differing types of Fixed Wing aircraft designs

3.1.1.1 Advantages & Disadvantages

One of the primary advantages of Lighter-than-Air craft is that no energy necessarily is required to keep the craft airborne, which drastically increases the potential flight-time of the craft. However, to lift a sensor payload and batteries a comparatively large envelope is required, having the side effect of giving the craft very high parasitic drag, thus lowering the manoeuvring speed, responsiveness and tolerance to winds, gusts and turbulence.

3.1.2 Fixed Wing

Fixed Wing (FW) aircraft are the most common type of aircraft. Lift is achieved using the airflow from the forward motion of the aircraft passing over one or more wings which are fixed to the body of the craft. There are many varied designs of FW aircraft, ranging from differing wing designs (Figure 3.1a), number of wings (Figure 3.1b), and powered/un-powered (gliders) (Figure 3.1c).

3.1.2.1 Advantages & Disadvantages

FW aircraft have the advantages of being effective and efficient at travelling long distances and also more resilient to turbulence. The main drawback for surveillance and inspection purposes is that the aircraft requires forward speed to generate the

Figure 3.2: An example of a VTOL aircraft the Bell/Boeing Osprey

lift required to remain airborne. This means that the the aircraft is *usually* unable to hover and flying at slow speed raises the risk of aerodynamic stalling and impairs manoeuvrability. The exception being a small group of aircraft equipped with *Vertical Take-Off and Landing* (VTOL) functionality, which have the ability to vector the engine thrust. Examples include designs such as the BAe Harrier and the Bell/Boeing Osprey (see Figure3.2) which could be argued as being a rotary wing aircraft. The Osprey utilises a propeller driven propulsion system, which can be rotated through the lateral axis to either pull the aircraft forward or vertically to produce lift much like a helicopter.

3.1.3 Rotary Wing

Rotary Wing aircraft create lift through the constant motion of a rotor through the air. Unlike fixed wing, a rotary wing aircraft doesn't require forward motion through the air to maintain lift, allowing them to hover or even fly backwards. There are many differing types of rotary wing aircraft, depending on the placement of the rotors, number of rotors and method of propulsion. The most common form

is the classic single rotor helicopter design, however in recent years there has been a substantial shift in the hobbyist/robotics community towards multicopter designs. The reason for the shift is twofold, firstly multicopters are mechanically simpler and secondly allow for easier payload placement.

3.1.3.1 Helicopters

The primary method used to control helicopters in-flight is through varying the pitch of the rotor blades as they rotate. The total thrust produced by the rotors can be varied by either slowing or speeding up the rotation or by varying the pitch of the rotor as it spins. On larger remote controlled helicopters or full sized helicopters the rate of rotation is generally kept constant and rely solely on varying the pitch whilst in-flight. Similarly, roll and pitch is controlled by changing the thrust produced by the rotors at different stages of its rotation.

To achieve this the rotors use what is called a swashplate (Figure 3.3), the swashplate is composed two elements. One which is linked with the rotor blades and therefore is spinning, and a non-rotating element which is connected to the controls or servos. The whole swashplate can be tilted in the roll and pitch axis through the non-spinning element, and effectively functions as a guide for the pitch of the rotors. Due to this, helicopters tend to be mechanically complex with many safety critical linkages, most of which moving and being adjusted at high-speed.

3.1.3.2 Co-Axial

A variant of the helicopter is the co-axial helicopter, where two in-line counter rotating rotors are used to generate lift (see Figure 3.4). Due to the rotors being counter rotating a tail rotor is not necessary as each rotor counters the torque generated by the other rotor. Co-axial rotors are however rarely used for full scale helicopters, partly due to the mechanical complexity and risk of the rotors colliding but also due to the high parasitic drag induced when travelling at speed [10].

Figure 3.3: Swash plate and linkages on a model remote controlled helicopter (rotor blades removed).

For small scale remote controlled helicopters however, co-axial helicopters are a popular design particularly for small low-cost *Remote Control* (RC) helicopters. With the modification of gyroscopically stabilising the top rotor through a flybar, and with the controls only influencing the bottom rotor, a very stable and easy to fly model can be achieved. The top rotor acts as a passive stabiliser opposing any induced motion or rotation (apart from yaw). By not having to supply linkages to the upper rotor the mechanical controls are no more complicated than a conventional helicopter. However, they can be stable and sluggish to fly, has a severely limited forward speed and the top rotor can easily collide with the bottom rotor in gusty conditions.

Figure 3.4: A full size co-axial helicopter (Kamov Ka-50)[11]

3.1.3.3 Multicopters

Multicopters refer to a type of helicopter which has three or more rotors, each placed on outriggers away from the centre of the helicopter, an example of which is shown

Figure 3.5: Example of multicopter design (Hexacopter)

in Figure 3.5. As the rotors are placed away from the centre of gravity, multicopters can control the pitch and roll of the aircraft by varying the total thrust from each rotor. This negates the need for a swashplate and the associated linkages, as the thrust from each rotor can be adjusted by simply varying the speed of the motor. The inherently unstable hover characteristics of the helicopter are also present in the multicopters and to some extent worsened. This is due to each rotor being powered by a separate motor, each of the motors will have a slightly different efficiency and the motor speeds will have to be rapidly changed to correct for any in-flight disturbance or commanded movement [12]. Just as the helicopter requires mechanical complexity to achieve flight, the multicopters rely on advanced electronics and sensors. Multicopters rely on a three axis rate gyroscope to sense if it is in the required pose. If not it individually varies the thrust produced by each rotor, resulting in a mechanically simple but electronically complex setup which is less fragile and requires less maintenance.

An added benefit of having the rotors placed away from the centre of gravity, is easy sensor mounting, unlike conventional helicopters where heavy payloads need to be mounted directly underneath the rotor. Multicopters have a large area in the

middle of the craft, where sensors can easily and safely be attached, both above and below the rotor line without the problems of affecting the craft's centre of gravity.

3.1.3.4 Advantages & Disadvantages

Table 3.1 highlights the advantages and disadvantages between the different types of rotary craft.

Characteristic	Helicopter	Co-Axial	Multicopter
Mechanical Complexity	High, swashplate and associated linkages & tail rotor.	High, one or two swashplates and associated linkages.	Low, fixed pitch rotor directly driven by an electric motor.
Electronic Complexity	None needed. Generally yaw is electronically stabilised.	None needed. Generally yaw is electronically stabilised.	High. Roll, pitch and yaw is electronically stabilised and mixers are used to define each motor's speed.
Rotor Redundancy	None	None	Yes, if more than 4 rotors and payload within limits.
Immunity to turbulence/gusts	Medium, in the right circumstances it is possible to have the rotor contact the tail rotor boom.	Low due to risk of collision between a flybar stabilised upper and controlled lower rotor.	High
Sensor Mounting	CoG directly below rotor, mounting points limited for heavier payloads.	CoG directly below rotor, mounting points limited for heavier payloads.	CoG in centre of craft away from rotors, allowing easy sensor placement both above and below the rotorline.

Table 3.1: Comparison between Helicopter, Co-Axial and Multicopters

3.2 Sensor Systems

Choosing the correct sensors is very important, as the sensors provide the only means for the robot to perceive the environment it is in. There are many differing types of sensors, ranging from contact to contact-less and active to passive. All sensors have strengths and weaknesses with what they can detect, along side with the quality of data they produce. Usually robots use several sensors to try and minimise errors, increase reaction times or detect what may be hidden from a single individual sensor.

Some of the big challenges with building UAVs are caused by the UAVs low payload weight capacity and small physical size restrictions, leading to large compromises over the number and type of sensors that can be used. The following section gives an overview of the current sensor technologies available and how they work. By knowing how the technology functions their strengths and weaknesses can be determined with regards to a UAV application.

3.2.1 Active Sensors

Active sensors function by emitting some form of radiation or energy which reflects or interacts with the environment which is then detected by the sensor. The majority of active sensors are used for measuring distance to an object, this can be achieved either through measuring the time of flight or through triangulation. Below are some of the most commonly used active sensors within the robotics community.

3.2.1.1 Ultrasonic range finders

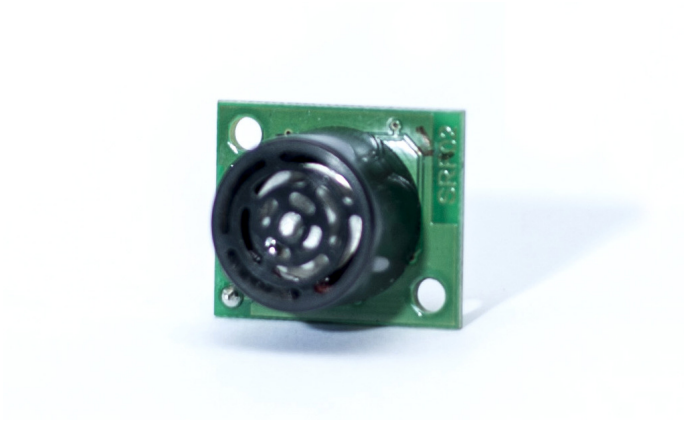


Figure 3.6: A typical ultrasonic range finder (SRF02)

Ultrasonic sensors (sometimes referred to as SoNAR) are very popular in robotics due to their low cost and low weight. The sensors function by periodically emitting an acoustic pulse (chirp) in the ultrasonic frequency range (∼40kHz and up). The pulse travels through the air until it is reflected by a denser object. By knowing the speed of sound through the material, in this case air, it is possible to calculate the distance travelled by the pulse and therefore the distance to the object.

Figure 3.6 shows typical ultrasonic range finder used on the Warwick Mobile Robotics robots. The SRF02 has an effective range from 10cm to 7m, and like most ultrasonic range finders it has a high field of view (approximately 20°). This makes ultrasonic range finders a good choice for detecting of objects, however poor at determining its precise position. The SRF02 has a resolution of approximately 1cm, the accuracy varies slightly with temperature, humidity and air pressure as the built in electronics computes the distances using the standard atmospheric model. Other larger units are available which can for-example account for changes of density due to temperature.

There are a couple of limitations which are prevalent with ultrasonic range finders:-

- Echoing occurs on every chirp and limits the possible update rate of the sensor.

If the update rate is too high a second chirp may be emitted before the original chirp has dissipated (reflecting off the walls in the room) and may misinterpret the two signals. Practically, with this particular sensor the effect is not noticeable under 10Hz.

- If the object has flat surfaces with an acute angle from the sensor it is possible that the chirp will reflect away from the sensor resulting in erroneous values or not detecting the object, instead accidentally detecting secondary reflected signals.
- Is susceptible to high frequency noise pollution.

3.2.1.2 Infrared Range Finder

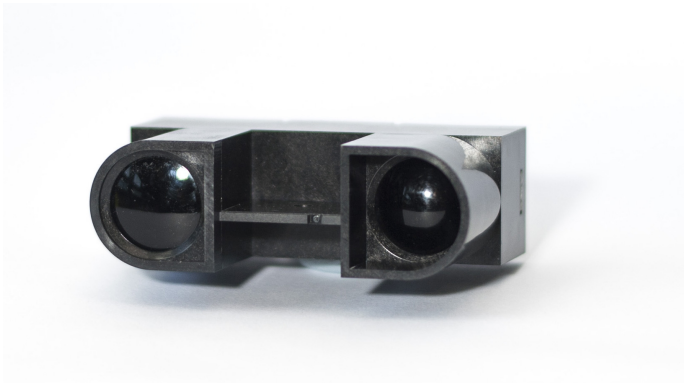


Figure 3.7: A typical infrared range finder (Sharp GP2Y0A700K)

Infrared (IR) range finders, much like the ultrasonic range finders offer a low cost, low weight measurement solution. Instead of relying on time of flight to calculate distances the sensors use triangulation. Figure 3.7 shows a typical sensor, the sensor comprises of two parts, the emitter, which is usually a infrared LED and a detector. The detector comprises of a lens and a linear *Charged Coupled Device* (CCD), a two-dimensional version of which is commonly used as the sensors in digital cameras. The emitter and detector is placed a set distance apart, therefore by varying the

distance from the object you vary the angle the light returns to the detector, this can then be measured and a range computed.

One of the main limitations of the sensor however is its relatively limited acceptable range, which tends to be fairly short distance. The sensor shown in figure 3.7 is a longer range version and has an effective range from 1m to 5m, outputted as an analogue voltage. One of the limitations of the IR range sensors is that it computes the distance from the strongest return on the CCD. This assumes the walls are plain and not textured and with ideally a white finish.

3.2.1.3 LiDAR/RaDAR

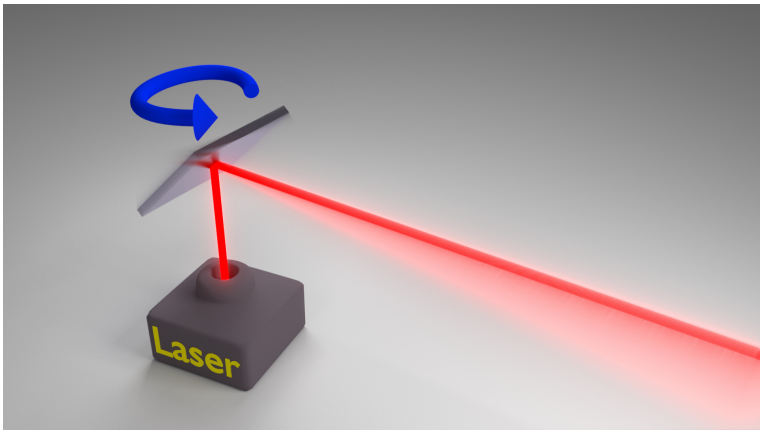
Figure 3.8: A small LiDAR Scanner (Hokuyo UTM-30LX)

Laser range finders are one-dimensional measuring tools. It operates through emitting a pulse of light in the form of a laser beam, and timing its time-of-flight to the object and subsequent reflection. This has several advantages over the infrared range finders (see 3.2.1.2). Firstly, as a laser is used, higher resolution measurements of the surroundings can be achieved due to the smaller beam width (which doesn't disperse to the same extent). Secondly, laser range finders accept a wider range

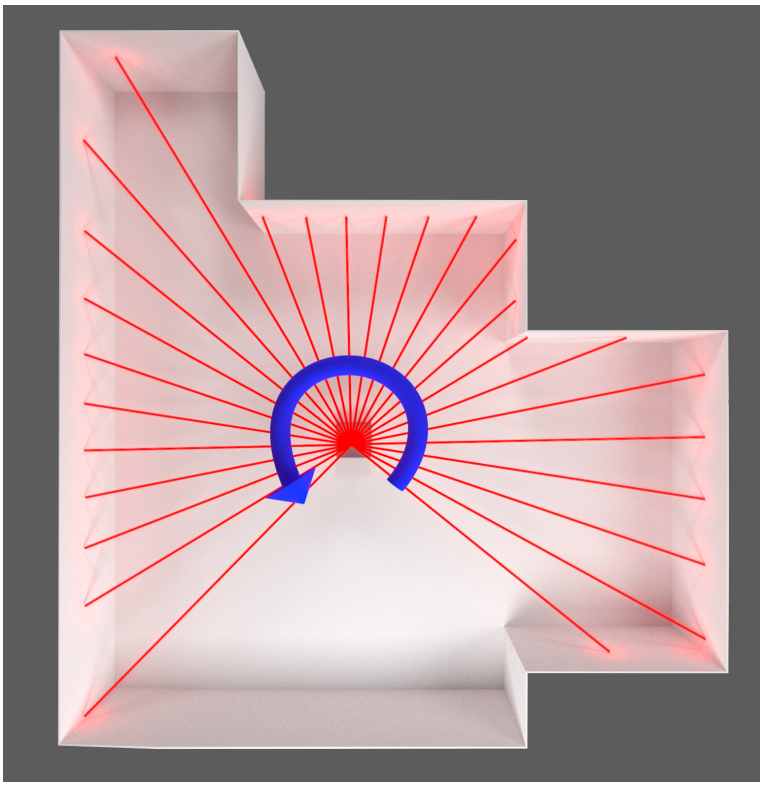
of distance measurement and are more accurate, which tends to be limited to the brightness of the laser and sensitivity of the detector and not the width and angular resolution of the device as with infrared range finders. The main disadvantage being mainly cost.

Laser range finders, however, can only measure the distance to the point where the laser is aimed. LiDAR (*Light Detection and Ranging*) scanners (Figure 3.8) operate using a laser range finder and collect two or even three dimensional data by shining the laser onto a spinning mirror, as shown in 3.9a. By scanning the laser and taking multiple measurements during the rotation (see Figure 3.9b), a detailed point-cloud representation of the room can be calculated. Some scanners are also able to either scan the laser vertically or utilise multiple laser beams and therefore be able to provide three-dimensional data.

As the LiDAR scanner is an optical sensor, it has several weaknesses. Firstly if the scanner is being used in a room with reflective surfaces such as glass, there is a high likelihood the laser will be reflected and the reflected distance being measured erroneously. Secondly, if the environment has significant amounts of dust, steam or other airborne particles, the effectiveness and reliability of the measurements will also be compromised[13].



(a) Diagram showing the sensor, laser and spinning mirror of a LiDAR scanner



(b) Diagram showing a single sweep of a LiDAR scanner

Figure 3.9: Diagrams showing the function of a LiDAR scanner

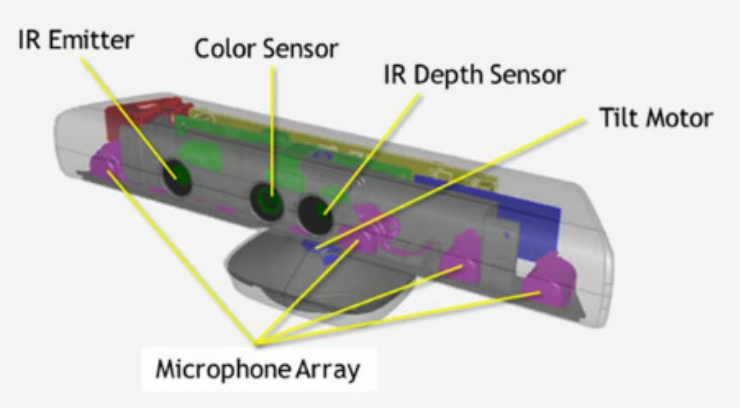
3.2.1.4 Flash LiDAR and Structured Light

Building on the principle of the LiDAR scanner, there is a type of sensor called a *flash* LiDAR (Figure 3.10). Its operation is more akin to a camera than a LiDAR scanner, a diffused laser is flashed forwards from the sensor illuminating the view of the sensor array. Each pixel then calculates the time of flight of the laser flash to hit the object and return, a three dimensional image can then be created by combining the measurements of all the pixels on the array [14]. Although a flash LiDAR generates three-dimensional representations of its environment, it is limited much like a camera in respect of its field of view being limited. Therefore to generate a scan of a room several scans need to be taken and stitched into a panorama.

Figure 3.10: A SR4000 flash LiDAR built by Mesa Imaging [14]

A similar approach to creating three dimensional imagery is using structured light to project a pattern grid over the scene, which a camera can detect and analyse. One such sensor is the popular Microsoft Kinect (Figure 3.11a), which projects a static infra-red reference grid onto the objects it is facing (Figure 3.11b). By knowing the grid's pattern and having the detector offset from the pattern projector it is possible to generate a depth (distance) map of the image. A big disadvantage with structured light based sensors is that they have a very limited range where they produce reliable and valid depth measurements, too close and the detector cannot

reliably distinguish the pattern, too far and the angular change is too small and the measurements become noisy (much like the IR range finder see section 3.2.1.2).



(a) Internals of the Kinect [15]

(b) Light pattern created by the Kinect [16]

Figure 3.11: The Microsoft Kinect sensor

3.2.2 Passive Sensors

Passive sensors function through purely observing the environment and do not emit any form of radiation or energy to collect their measurements.

3.2.2.1 Cameras

We heavily rely on vision to interact and navigate with the world we live in, with the advancement both in camera equipment and image processing, the use of cameras is

becoming more commonly found in robotics. There are many possible applications of a camera sensor, from simply getting a raw image which could be returned to the operator, to more advanced features such as object detection and tracking, visual odometry (much like an optical mouse) and even three dimensional depth extraction giving similar data to that of the LiDAR.

Unlike humans, cameras don't have to operate in the visual spectrum. Specialised cameras can for-instance detect heat (infrared) or ultra-violet radiation (higher and lower frequencies are possible to image, however building capable detectors and lenses become increasingly difficult).

3.2.2.2 Inertial Measurement Unit (IMU)

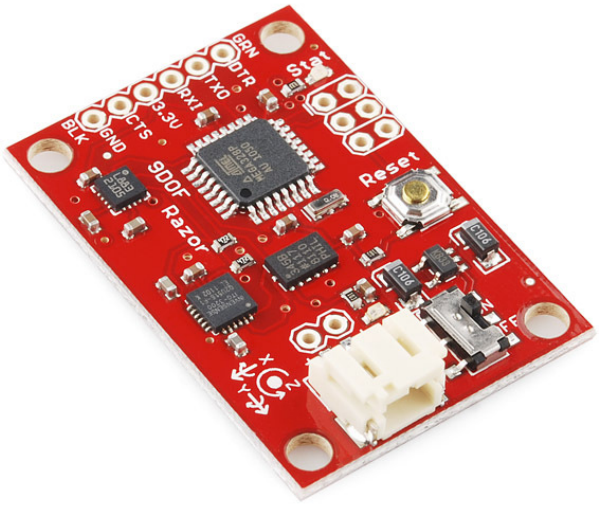


Figure 3.12: A small electronic IMU[17]

Inertial Measurement Units (IMU) detect and measure changes in the angular and linear velocity of the object to which they are mounted and are very commonly used to estimate the orientation of a robot, an example of which is shown in figure 3.12.

The robots velocity and position can be computed through respectively integrating or double integrating the detected accelerations. This leads to one of the primary limitations of the IMU - drift. All sensors have noise or bias and if for instance the robot's position is being calculated, any erroneous data would also be double integrated and lead to large errors accumulating over time.

For angular measurements drift can be lowered through using the data from multiple sources using the assumption that the average detected linear acceleration will be gravity. Using this assumption the roll and pitch axis can be aligned with the horizon, given that the robot does not accelerate laterally in one direction for an extended period of time. Detecting heading drift is more difficult and requires the use of magnetometers which function much like a compass. A three dimensional magnetometer can also help to stabilise the roll and pitch axis, but much like any compass suffers greatly from electromagnetic interference from for-instance electric motors.

Traditionally IMUs have utilised mechanical gyroscopes in applications such as aviation INS (Inertial Navigation System) units prior to the adoption of GPS (discussed in more detail in Section 3.3.2.1). The more modern units using fibre optic technologies enabling higher accuracy without moving parts. These units however weigh several kilograms and are expensive.

Recent breakthroughs in "solid state" MEMS technology has allowed accelerometers to be fabricated "on chip" (figure 3.13) at a fraction of the cost and weight, with only a small decrease in accuracy.

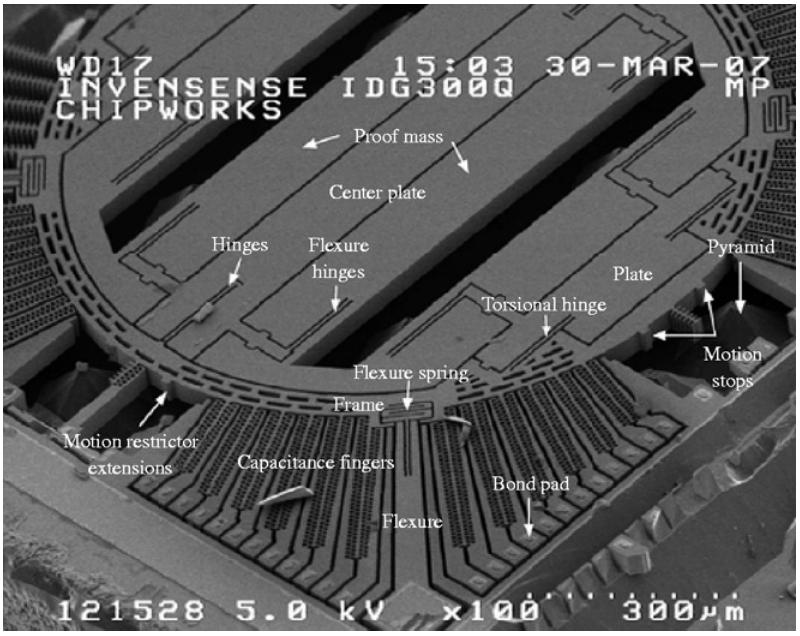


Figure 3.13: The silicon substrate of a gyroscopic rate sensor seen using a scanning electron microscope[18]

3.2.2.3 Pressure Sensors

Pressure sensors are very common in aviation and their primary means is to measure the speed and altitude on larger aircraft. Atmospheric ambient pressure decreases with altitude at the rate of approximately 1hPa per 28ft (8.5m) below ∼2000ft (610m) above mean sea-level[19]. Through comparing the ambient (static) pressure with a known reference pressure the aircraft's altitude can be determined. Airspeed can be measured through comparing the pressure from a forwards facing pitot tube with the ambient static pressure, yielding primarily the velocity driven dynamic pressure [20].

Small electronic pressure sensors which are suitable for model aircraft can be manufactured using MEMS technology (see Figure 3.14). In the case of using a pressure sensor as an altimeter there are limitations to its use, such as small (<10cm) motions being difficult to detect due to the small variation in air-pressure. Also any turbulence or wind buffeting will cause erroneous measurements.

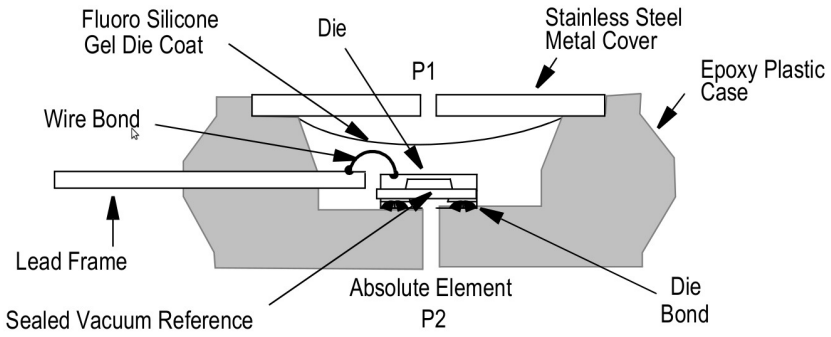


Figure 3.14: The internals of a MPX4115 static pressure sensor[21]

3.2.2.4 Radiation Sensors

The ability to be able to sense radiation is an integral part to performing classification of a nuclear structure, allowing anomalies to be detected and determining the dose that the workforce would be exposed to if they entered. Dose signifies the accumulated exposure to radiation, a dosimeter may or may not be able to measure the immediate exposure to radiation, but it is used to determine the type and amount of radiation it has been exposed to, essential for long-time worker radiation monitoring and safety.

There are a multitude of different technologies which enable the detection of ionising radiation, the four primary types are listed below:-

Gaseous Ionisation Detectors function on the principal that an enclosed volume of gas (commonly argon or helium) when exposed to ionising radiation, will emit ions which can be detected using measurement circuitry. There are three primary types of gaseous ionisation detectors, depending on the voltage applied to the anode and cathode within the chamber. ***Ion chambers*** operate at the lowest voltage (region II in Figure 3.15), where all the detected gas ions are created purely through ionisation events, which allows for accurate dose measurement, however require sophisticated electronics to measure the low signals produced (especially for gamma radiation). The ***proportional counter*** utilises a slightly higher field strength than used for the

ion chamber detector (region III in Figure 3.15). In this region the ions created by each ionisation event are accelerated by the electric field to an energy higher than that of the ionisation event, allowing for secondary ions to be produced through impact. The amount of secondary ions produced is *proportional* to the energy of the incoming radiation, therefore not only can the type of radiation (alpha, beta or gamma) be distinguished, but also its energy.

Geiger counters operate in the much higher field strength - Geiger-Müller region (region V in Figure 3.15). In this region not only are there secondary ions created like the proportional counter, but the gas ion has enough energy to produce an *Ultra-Violet* (UV) photon. This UV photon has enough energy to ionise other gas molecules, causing further avalanches. Geiger counters have the advantage that a single ionisation event causes an easier to detect "maximum response" regardless of power or type of the incoming radiation. This leads to the major disadvantages with Geiger counters, namely that they cannot directly discriminate between the power and type of the incoming radiation [22].

Semiconductor Radiation Detectors function much in the same way as a gaseous ion chamber, however instead of using gas as a detection medium a p − n or p − i − n semiconductor junction is used. When an ionising particle enters the intrinsic region, it will lift a certain amount of electrons from the valence band into the conduction band, the electron/hole pair are then moved to the $n+$ and $p+$ regions respectively. The ionisation event can then be measured as a small current generated by the p − n junction. Benefits being that they are more robust being a solid-state device and that it is possible to measure the energy and type of radiation that it is exposed to [22].

Scintillation based Radiation Dosimeters function on the principal that some inorganic crystalline, and organic materials, emit photons when absorbing ionised radiation. The energy and type of radiation can be determined through the characteristics of the emitted photon, which is commonly amplified using photo-multipliers

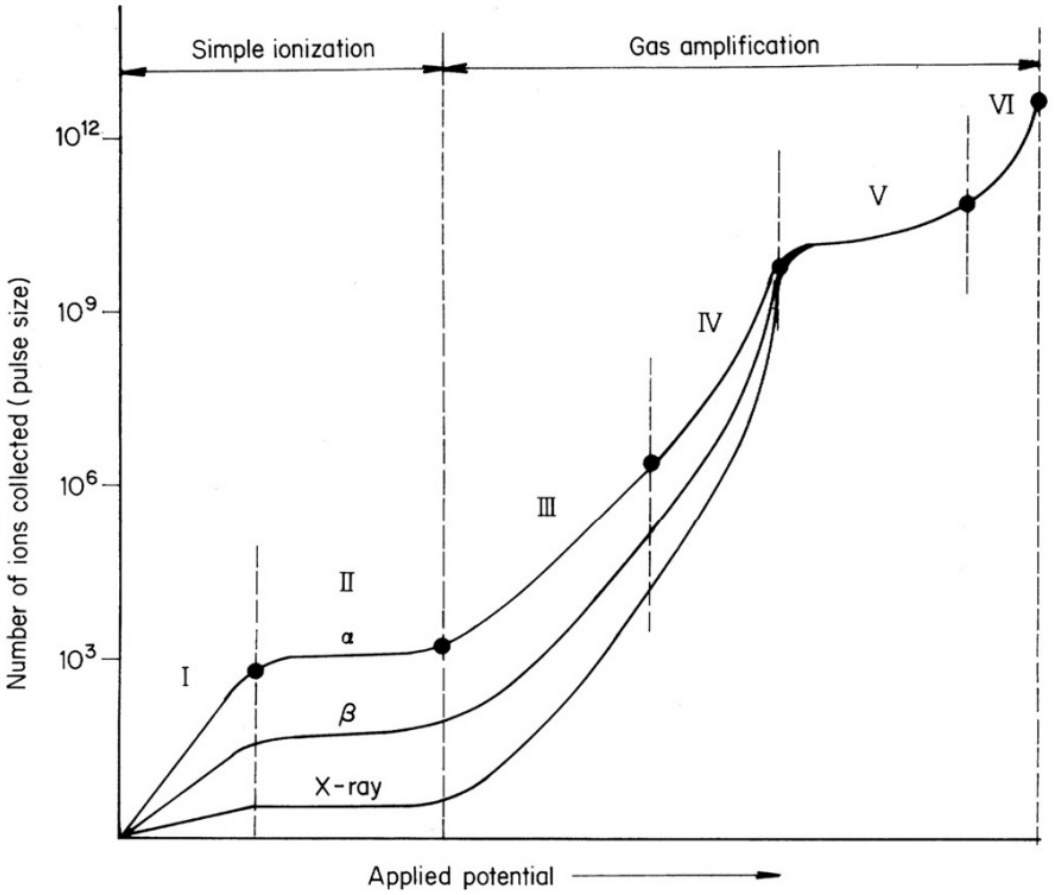


Figure 3.15: Relationship between produced pulse size and the potential applied to the electrodes in a gas ionisation chamber.
Region I - recombination region; Region II - simple ionisation region; Region III - proportional region; Region IV - limited proportional region; Region V - Geiger-Müller region; Region VI - continuous discharge region [22].

for easier detection. Each material has a differing response when being ionised, so through knowledge of the radiation being measured and intelligent choice of detector material accurate and effective detectors can be built [22].

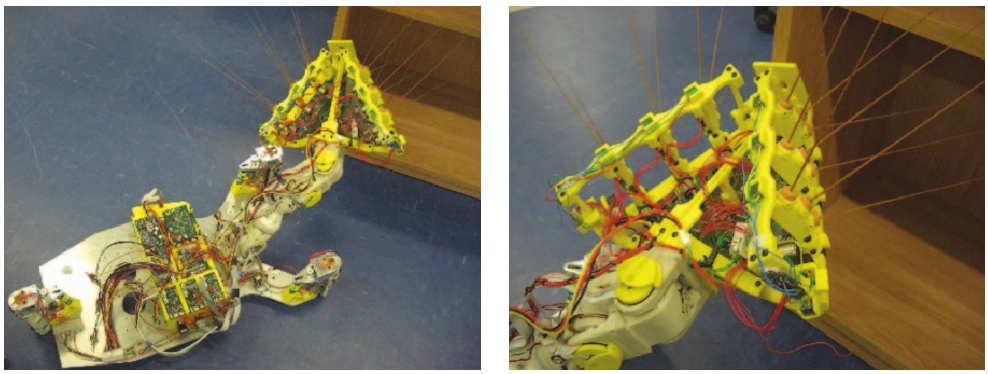
Film Dosimeters function on the simple principal that photographic film blackens when it is exposed to ionising radiation. By overlaying metallic patterns of varying thicknesses, it is possible to determine the type of (alpha, beta or gamma) radiation the film has been exposed to. Film is a popular choice for dosimetry, as it is low cost and simple. The disadvantage is that there is no instantaneous notification of exposure, the radiation levels can only be determined once the film is developed [22].

3.2.3 Contact Sensors

Although not generally appropriate for flying vehicles, contact sensors are included for completeness. Contact sensors function through touching the object they are measuring unlike measuring an electromagnetic response such as the passive and active sensors above. They are used extensively for collision avoidance as they are impervious to problems found with active and passive sensors such as dust and reflections and therefore guarantee detection of any physical object within range. An example of a contact sensor are whiskers, used for-instance on SCRATCHbot developed by Bristol Robotics Laboratory (Figure 3.16) [23].

3.3 Localisation

Accurate and reliable knowledge about the robots pose and position in its environment is the key to successful autonomous control and planning. Without this knowledge, the robot would be unaware of where to travel in order for it to reach its waypoint or to correct itself if it has drifted off-course. There are two primary



(a) SCRATCHbot robot[23] (b) SCRATCHbot Closeup[23]

Figure 3.16: SCRATCHbot robot developed by Bristol Robotic Laboratory

methods for a robot to determine its position in space "Aided Localisation", relying on external devices (see 3.3.1), or "Autonomous Localisation", relying purely on on-board sensors (see 3.3.2).

3.3.1 Aided Localisation

Aided localisation is where the robot relies on external transmitters or markers to determine its position within the environment. There are many commercial products and techniques available depending on the accuracy required, type of environment and cost, a few of which are listed below.

3.3.1.1 Global Positioning System (GPS)

Figure 3.17: An EM-406A SiRF III GPS Receiver[17]

An extremely popular method for outdoor localisation is through using the *Global Positioning System* (GPS). GPS is built up from a constellation of 24 satellites in a polar orbit ∼20200km above the earth, with each satellite broadcasting its current position and time. By capturing this data from three or more satellites simultaneously, the receiver is able to calculate the distance to each satellite, through knowing the position of the satellites the receivers position can then be calculated [24].

The GPS receivers have the added benefit of being very cost effective, compact and lightweight, some of which only weighing tens of grams including both detector circuitry and antenna (see figure 3.17). Typically an absolute accuracy of approximately 5 metres can be expected (greater with a partially obscured sky), but is able to detect much smaller movements, used to detect velocity and heading, obtaining a typical update rate of $1-4$Hz . Higher accuracy can be achieved but requires the use of very expensive, larger and heavier surveyor grade receivers [25]. The

drawbacks are however, that it can only function outdoors and when the receiver is in line of sight of at least three of the satellites. There are situations, even when outdoors, where a building or structure may obscure the view of the satellites, causing degraded accuracy or erroneous position updates through either signal loss or multi-path propagation. This situation is referred to as an urban canyon [26], and can cause severe GPS disruption when operating in built up areas.

3.3.1.2 Triangulation and Radio Beacons

In aviation, prior to the deployment and acceptance of GPS, radio beacons were the only method to get the aircraft's absolute location when navigating out of sight of the ground. The first beacons were simple AM radio transmitters known as *Non-Directional Radio Beacons* (NDBs). The antenna on-board the aircraft could detect the direction in which the signal was strongest and the instrument would give a relative bearing ("point towards") the direction of the beacon [27].

Due to the low accuracy of the bearing obtained from the NDB and that various environmental effects cause the signal to appear to be coming from slightly different directions, another more sophisticated mode of localisation was devised. These newer beacons, *Very High Frequency Omni-directional Radio Beacons* (VORs), transmit not only an AM signal but also a similar FM signal the phase of which is proportional to the compass rose of the beacon [27].

Instead of determining the aircraft's bearing to the beacon, VORs operate in reverse, the aircraft instead determines its location based on its orientation with regards *to* the beacon. To determine the aircraft's location two or more VORs are used, triangulating the position using the bearing angles. Some VOR beacons are also fitted with *Distance Measurement Equipment* (DME) which allows the aircraft to measure its distance from the beacon. This has the benefit that only one beacon is needed to determine the aircraft's position (as the bearing and distance from the beacon are both known) [27].

Higher accuracy, more localised positioning solutions are available using similar principles and relying on external beacons to localise the detector. An example of which is the XSens MVN MotionGrid which utilises a constantly transmitting tag. The transmissions are monitored by a minimum of nine detector "beacons" with an accuracy of approximately 5 – 8cm within a 20x20metre work area. The work area can be expanded using additional beacons or at the cost of reduced accuracy [28].

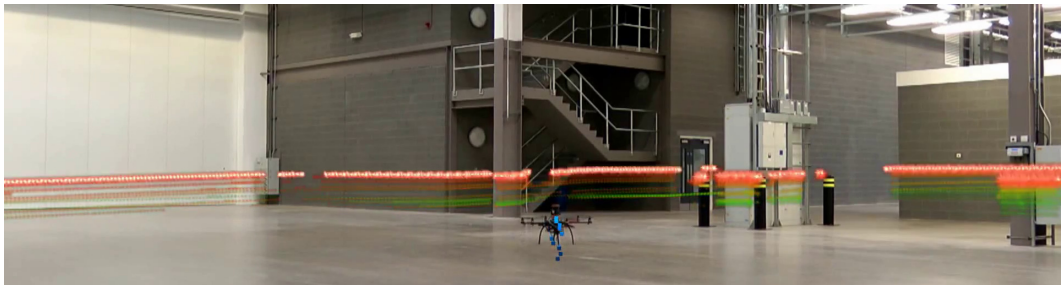
3.3.2 Autonomous Localisation

Autonomous localisation is the opposite to assisted localisation, as it does not require off-board devices for the robot or craft to over localise itself, thus relying solely on its own sensor data to extract the position.

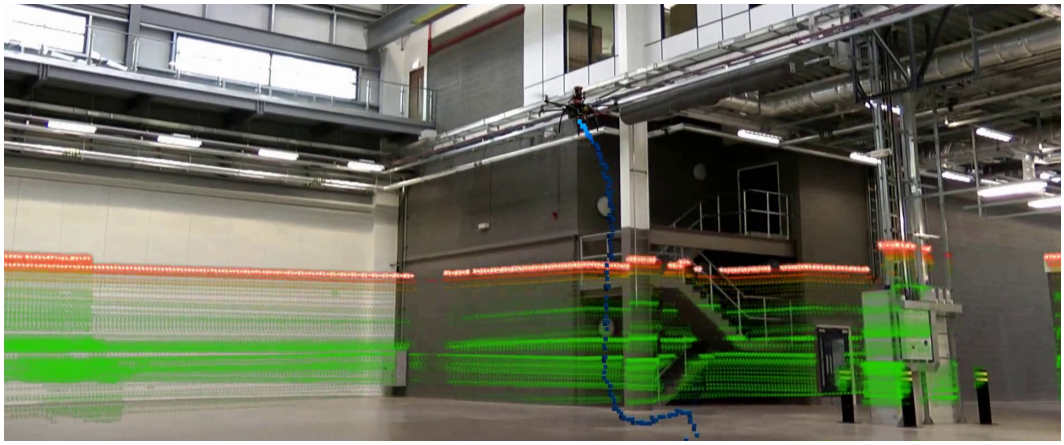
3.3.2.1 Inertial Navigation System (INS)

Inertial Navigation Systems (INS) rely on double integration of the acceleration of the craft as measured by an IMU (see 3.2.2.2). The pitfall with INS is with regards to the double integral - any bias or erroneous measurement of the acceleration will cause an exponential divergence in position (*drift*). To counter this another localisation method has to be occasionally used to correct the drift, the frequency of which relies on how "good" the sensors are and the accurate requirements of the computed position.

Higher accuracy INS, such as the types fitted to for instance the Apollo space craft or the Concorde are both expensive and heavy, making them unsuitable for smaller robotic platforms. With the increasing quality and availability of solid state *Micro-ElectroMechanical Systems* (MEMS) accelerometers, INS has only the past decade become a viable and cost effective method for small scale robotics.



(a) Initial stages of map building



(b) After significant movement of the UAV

Figure 3.18: Visualisation of SLAM (Orange being recent data, green being the map being built and blue being the calculated position of the UAV). See Appendix D.2

3.3.2.2 Simultaneous Localisation and Mapping (SLAM)

Simultaneous Localisation and Mapping (SLAM), is the process of using on-board sensor data to build, in real-time, a representative model of the environment, while simultaneously using the same model to localise the robot. Much akin to drawing a map of a newly explored area and at the same time using the map to determine your current location as shown in figure 3.18.

It is said that the problem of SLAM has already been solved at the conceptual and theoretical level, however implementing with real-world "noisy" sensors, cluttered environments and operating with constrained processing power poses a significant challenge [29, 30].

Non-Probabilistic vs. Probabilistic There are two main branches to approach SLAM, either using a non-probabilistic approach which assumes that the sensors and the matching algorithm are ideal and have no error which (at present) is incorrect. Probabilistic methods attempt to overcome this shortfall by applying statistical methods to account for the inaccuracies in the sensor's collected data. Using the probabilistic method however adds significant processing overheads as each point is not only matched, but the likely-hood of its location is also computed.

For this reason probabilistic SLAM approaches tend to not use the raw sensor data, instead it relies on using landmarks or features extracted from the data. The positioning and probability calculations are then performed solely on these landmarks, using as few as possible as the computational requirements scale to the square of the number of landmarks [29].

Herein lies a significant problem and one can severely limit the environments the robot can operate in. Landmarks need to be efficiently extracted from the raw sensor data (possibly from multiple sources), common and easy to extract landmarks will vary between different applications and environments. For office and corridor style environments edges and corners are commonly used, and other algorithms exist which attempt to identify its own landmarks without relying on predetermined ones.

SLAM Variants Over the years a multitude of different algorithms have been developed that attempt to solve the difficult problem of SLAM, at present there is no one-unified algorithm which will work in all environments using any type of sensor, instead many research groups build more specific algorithms based on their operating environment and type of robot. A few of these variants are discussed below:

Orthogonal SLAM[31, 32] is based on the assumption that the environment consists of orthogonal planar surfaces, such as a tidy office style environment. By extracting the planes out of the raw pointcloud and then matching the features a significant

speed improvement can be found, as well as increased robustness due to the reduced chance of false positive matches (compared to using raw pointcloud data), however this method requires known and defined geometric features to function correctly.

FastSLAM[33, 34] is an example of a particle based (probabilistic) feature based SLAM algorithm. The algorithm also enables loop-closing which is invaluable if the robot is operating in cyclic environments whereby the robot may return to the start position through a previously unexplored route. Without loop-closing the accumulated errors usually cause the looping point not to line-up causing further errors as the robot explores further, however, with loop closing the robot identifies that it has returned to certain location and re-estimates the positions and uncertainties of the previous scans to attempt to align the two paths. It has been shown that the original FastSLAM algorithm degenerates over time[35]. There are also variants which utilise similar methods to FastSLAM but instead of relying on landmarks, function on raw laser range data[36, 37].

There also exists a number of minimalist SLAM algorithms such as *TinySLAM*[38] which offer basic probabilistic LiDAR based SLAM in under two hundred lines of C-language code, proving that SLAM doesn't require lengthy sophisticated algorithms to function.

Iterative Closest Point (ICP)[39, 40, 41] is a three-dimensional registration algorithm used to align two or more three-dimensional datasets such as point-clouds, features or geometry. The algorithm explained in further detail in 5.4.1 on page 90 was not developed specifically for the purpose of SLAM, however, it has significant advantages such as ease of implementation and that its principal operation is representation-independant. This means that it functions on any environmental representation and is not limited to a specific method such as raw point-cloud, landmark features or geometric representations, with the disadvantage that it solves to a local minima of the mean-square distance metric, meaning the algorithm is not guaranteed to find the global minima (correct alignment) of scans in all situations.

There are also a number of visual SLAM algorithms such as [42] which function

using feature detection from a single vision camera. Other algorithms also exist where an RGB-D sensor such as the kinect is used[43], offering both image and depth maps to aid the map building process.

Specific localisation algorithms used on related projects are discussed below in section 3.5 and later summarised in section 3.6 on page 55.

3.4 Autonomy

Autonomy can be considered the "holy grail" of mobile robotics, for ideally it is being capable of doing its job without intervention from people. This has been one of the forefronts in robotics research for the past decade, and has had many breakthroughs primarily for rigid tasks while operating in semi-structured environments. An example of which is self-driving cars, the task is rigid and simple, drive to the destination without crashing or endangering others. Doing this however, is more challenging, as the robot will need to be aware of the traffic laws, aware of other road users and be able to plan it's journey. What makes the task easier is that the road network is generally a structured environment, with high accuracy road maps and road markings are available for planning and guidance, along with one way and speed restriction data.

Autonomy is a collection of various systems functioning in synergy. The robot needs to be able to calculate where it is, where it is going and how to best get there. A task only made more difficult when further unknowns are introduced such as incomplete knowledge of the environment or a vague, potentially unreachable target position to find an object. Similar to the analogy of trying to get between two points in an unknown maze.

For anything but simple tasks, especially in commercial products, full/true autonomy is either too difficult, costly or simply not required. Instead partial autonomy is often implemented, mainly to aid the operator, improving the ease of use or speed of

Level	Types	Definition
Level 1	Fully Tele-operated	The robot is fully tele-operated, much like a remote controlled toy. All sensor data is returned to the operator and not used for any on-board functions.
Level 2	Assisted Tele-operation "Tele-Assist"	The robot is controlled by the operator much like Level 1, however the robot uses sensor data to aid the operator. This can be thought of as kinematics on a robot arm as opposed to joint control.
Level 3	Assisted Autonomy "Tele-Robot"	The robot is autonomous in most functions, only relying on the operator to verify correct operation or to decide on a route or path.
Level 4	Fully Autonomous	The robot is fully autonomous and does not require human participation at any point during the "mission".

Table 3.2: The Different Levels of Autonomy

operation as opposed to the novelty or completely independent system. The amount of autonomy used in a robotic system can be classified into four levels as shown in Table 3.2, ranging from a fully tele-operated robot where the user is in complete control and the robot has no autonomous features, to fully autonomous systems.

Small robotic UAVs have only become viable in the past couple of years due to the improvement and miniaturisation of sensor, computational and battery technologies. Previously, autonomous capability could only be achieved in the larger "full scale" UAVs, primarily developed by the military for surveillance missions, due to their cost and complexity. For these UAVs the military has focused heavily on the use of autonomy to reduce the operator's/pilot's workload and enable them to control several vehicles simultaneously, essentially shifting the pilot from flying the craft towards directing which targets to find, track or follow. Lately, this has become possible with smaller and cheaper UAV systems, enabling their use with hobbyists, photographers, police and military forces. With the widening availability of suitable

sensors, lightweight UAV platforms and powerful miniature computers, small UAVs are becoming a popular choice for robotics research as well as commercial platforms.

3.5 Examples of Related Projects

The following section aims to highlight and discuss the design choices made by similar UAV projects, along with their performance and limitations. With the hopes of learning from their work, integrating the best solutions, lessening the risk of re-inventing the wheel while avoiding common pitfalls.

3.5.1 International Aerial Robotics Competition

The *International Aerial Robotics Competition* (IARC) is organised by the *Association for Unmanned Vehicle Systems International* (AUVSI) and was started to help promote the academic development of autonomous flight using small UAVs. The competition started in 1991 and has primarily focused on outdoor UAV missions, such as autonomously finding an object and delivering it to a certain location. In 2009 (fifth mission) the focus of the competition turned towards autonomously navigating indoor or GPS-denied environments, mainly due to the increase in popularity of small robotic UAVs [44].

The fifth mission comprised of an indoor scenario, where a control panel with a blue flashing *Light Emitting Diode* (LED) had to be autonomously found and photographed with no human interaction. The environment was highly-structured and largely two dimensional, much like an room/corridor arrangement with minimal clutter, see Figure 3.19. In some areas of the arena windows were added to increase the difficulty of the task, as the UAV would have to locate the opening and then be able to fly accurately enough to pass through. The UAVs were not allowed any prior knowledge of the layout of the arena, instead having to explore, map and plan a route through the arena as it progressed.

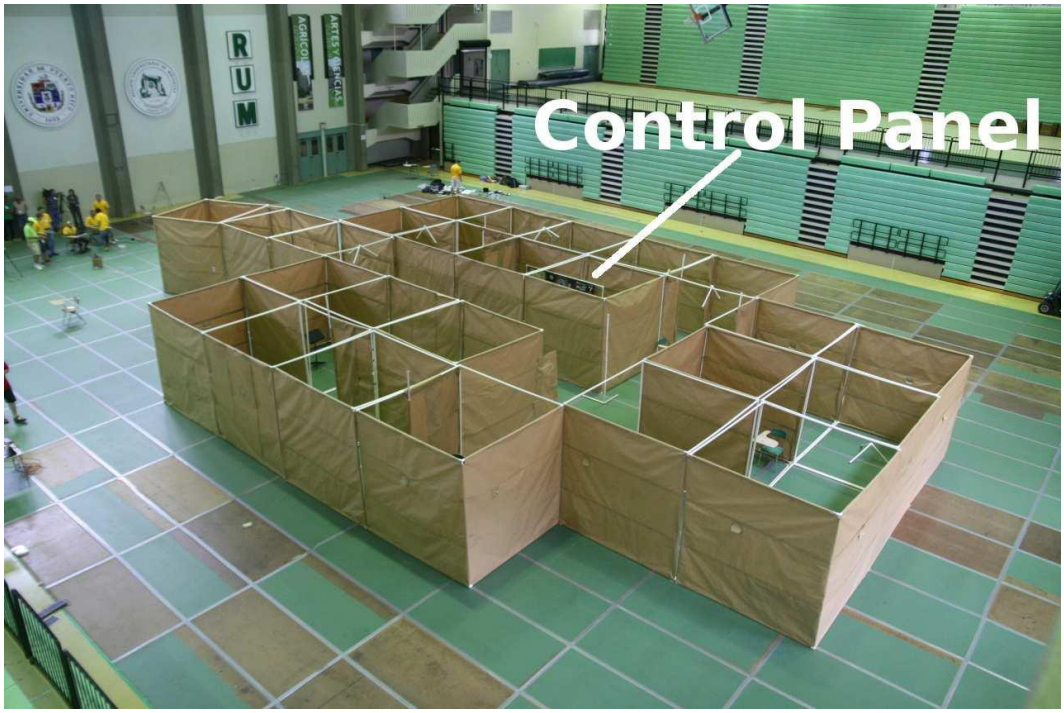


Figure 3.19: The IARC 2009 Arena[46]

A team from MIT, were the only participants who managed to complete the full mission, albeit on the last run of the competition, which demonstrated that UAVs were reaching the level of being capable of autonomous flight in structured yet unknown environments.

Since then, a sixth mission has been held in October 2012, with more difficult and demanding objectives. The task being to enter a similar yet more cluttered environment with the goal of finding and replacing a USB flash drive. Although showing promising results no teams completed the mission successfully and the competition has been re-scheduled for 2013 [45].

3.5.2 MIT-Ascending Technologies UAV

The *MIT-Ascending Technologies* (MIT-AscTec) were the winners of the 2009 IARC competition (see 3.5.1), managing to build a miniature UAV system, capable of flying in unknown yet semi-structured environments autonomously.

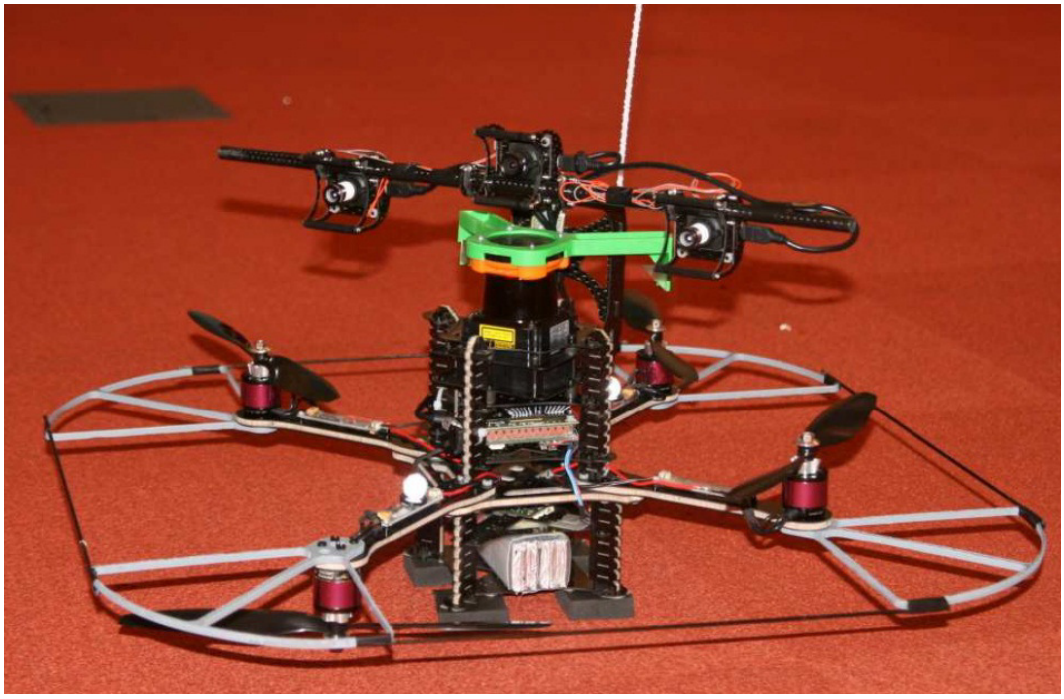


Figure 3.20: The MIT-AscTec UAV[47]

3.5.2.1 Flying Platform

A quadrotor was chosen as the robotic platform, due to their partnership with Ascending Technologies GmbH a prototype quadrotor was developed (Figure 3.20). Ascending Technolgies GmbH had several commercial-off-the-shelf quadrotors available, but all lacked the payload capacity needed to lift the sensor payload [46]. An additional modification to the quadrotor was the inversion of the front rotor, switching from a pulling to a pushing rotor, which increased the viewing angle of the camera system.

3.5.2.2 Sensors

The sensors on-board the MIT-AscTec were chosen so that the UAV would be capable of both LiDAR mapping and visual inspection. A Hokoyu UTM-30LX LiDAR was used which has an effective range of 30 metres, on the side of which a mirror was mounted to allow for accurate height measurement. Two video cameras

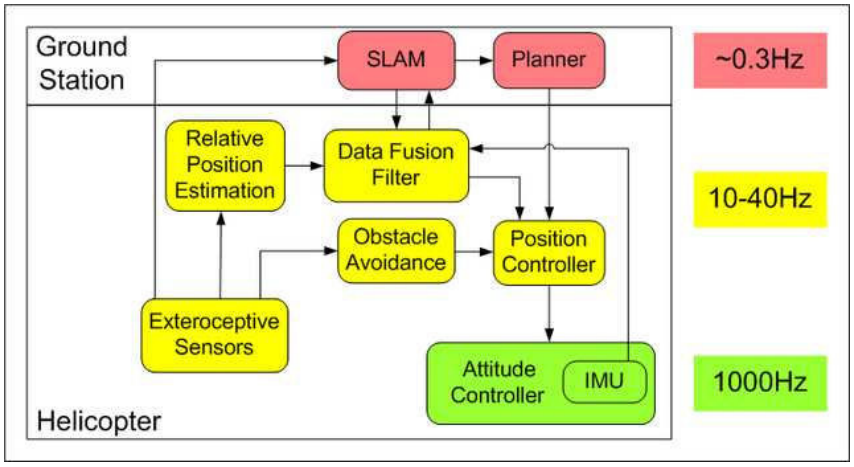


Figure 3.21: System overview of the MIT-AscTec UAV[47]

were also used to form a 3d camera with a wide interocular distance to give better medium/long-range depth perception [46, 47].

3.5.2.3 Localisation and Autonomy

To estimate its position in space the UAV uses a combination of visual and LiDAR based SLAM (GMapping[46, 36]). The height of the UAV is obtained through the use of a mirror mounted on the side of the LiDAR, which directs a portion of the scan perpendicularly below the craft. This allows for accurate height measurement up to the range of the LiDAR scanner. However, this technique requires precise knowledge of the UAV's roll and pitch to correct for any off-axis measurements and it is necessary to filter the data to remove sporadic returns when passing over objects.

The developed SLAM and planning algorithms require large amounts of processing power and memory, more than is available using (with sufficient speed) the on-board computer. Figure 3.21 shows a simplified diagram of the developed system. The key to solving the processing requirements for the SLAM algorithm was to process the data off-board on the ground station where powerful computers could be used. The drawback being that it takes a certain amount of time to relay the sensor information

to the ground-station, compute the position, plan the next move and then return the data to the UAV. It was discovered that the processing time (approximately 3 seconds) was too great to correct for the lateral dynamics of the UAV. Instead the UAV uses a "Relative Position Estimator" which acquires the current sensor data and estimates the motion of the UAV, much like an odometer. Given the lower complexity of the algorithm it can be run on-board and stabilise the UAVs motion between position and target updates from the ground station.

Due to the competition regulations of IARC the MIT-AscTec UAV was designed to be fully autonomous, having the ability to fly in a one story building environment and also capable of finding the simulated door openings and windows to progress between rooms. In reality, this is more difficult as the robot has to be able to detect if the window is actually open (as glass is transparent/reflective for LiDAR scanners).

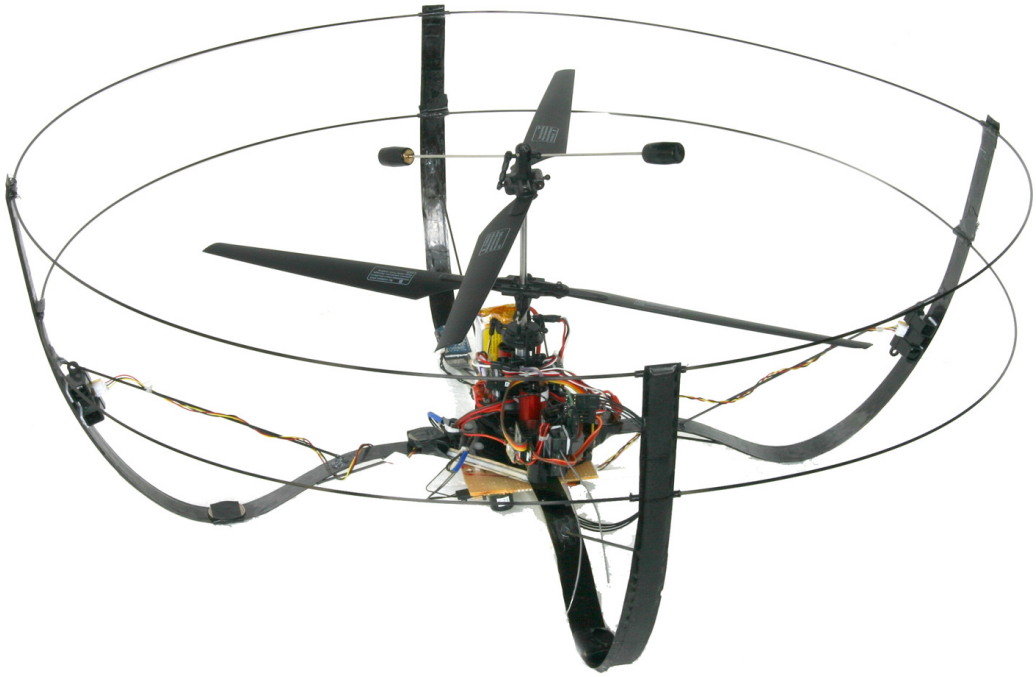
3.5.3 Georgia Tech Aerial Robotics

Much like the MIT-AscTec team (3.5.2) the Georgia Tech Aerial Robotics team also took part in the 2009 International Aerial Robotics Competition. Although they did not manage to successfully fully complete the course, their approach and solution was notable.

3.5.3.1 Flying Platform

The approach taken by the Georgia Tech Aerial Robotics team for the 2009 IARC competition was to base their robot on a "passively" stable rotary wing design. One such design is a co-axial helicopter whereby the top rotor, through a flybar mechanism, partially counteracts the UAVs motion. The team based their UAV on a commercial-off-the-shelf E-Sky Big Llama shown in figure 3.22a, modifying it through removing the bodywork, attaching sensors, upgrading to more powerful motors and adding a rotorguard shown in figure 3.22b [48].

(a) A E-Sky Big Llama (off-the-shelf) co-axial helicopter[48]



(b) The Georgia Tech Aerial Robotics UAV[48]

Figure 3.22: Georgia Tech Aerial Robotics UAV, as purchased (a) and as finished (b).

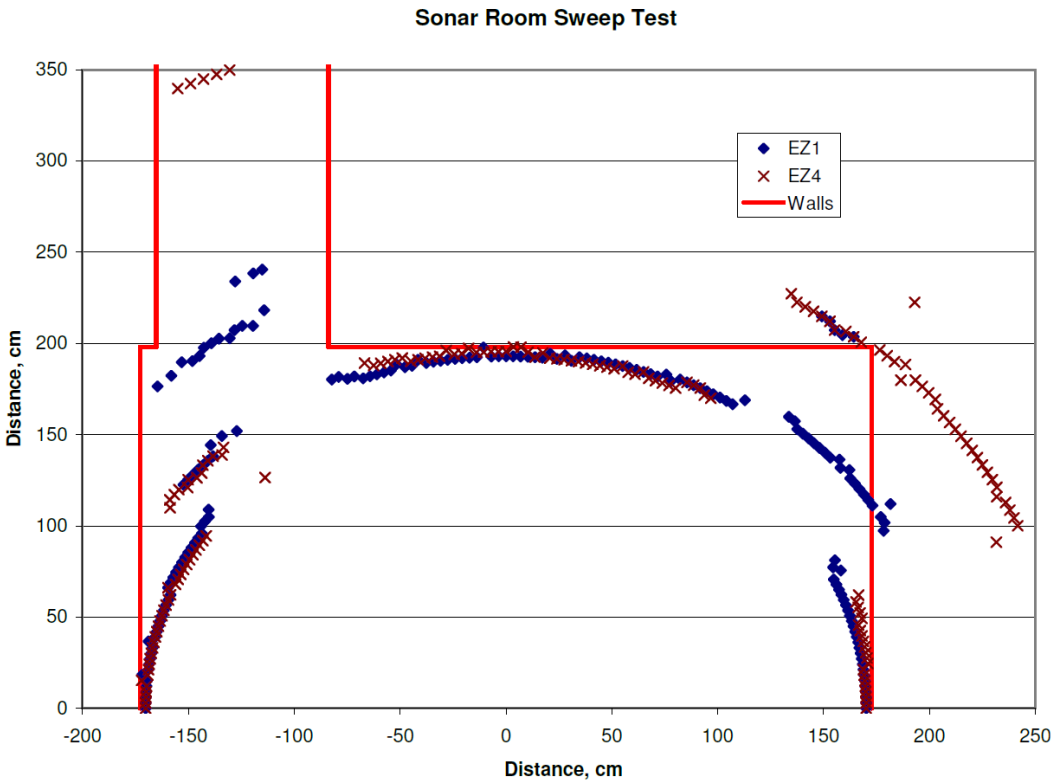


Figure 3.23: Results of scanning an ultrasonic range finder through 180 degrees inside a room. EZ1 and EZ4 representing the model numbers of the ultrasonic range finders used [48].

3.5.3.2 Sensors

Due to the limited payload capacity of the UAV platform the team focused their efforts on using only low-cost, low-weight sensors. This ruled out the use of LiDAR scanners due to their cost. Instead the team used a combination of ultrasonic and infrared range finders. An ultrasonic range finder was used to measure the height of the UAV. The reason being that the sensor is unaffected by small pitch and roll angles due to its large spread, whereas the infrared and laser sensors would over-read by not measuring the height directly under the craft. Initially, ultrasonic range finders were also used for estimating the lateral position of the UAV and mapping. However, tests showed that it was difficult to distinguish objects, walls and doorways (Figure 3.23). Instead the team favoured the use of infrared range finders, although, with their limited acceptable range inputs, this still proved a more sensible sensor choice [48]. A camera was also fitted to the craft, its sole purpose being to detect the blue LED on the control panel[49].

3.5.3.3 Localisation and Autonomy

The UAV was not capable of detailed mapping due primarily to the low angular resolution (low data-density) of the chosen sensors (see Figure 3.23). Instead the UAV used a "wall following" technique to navigate the maze to attempt to find the control panel[49]. If it detected that it was in a room, it would try to centre itself in the middle of the room, then spin through 360° in order to scan for open doorways. Using this method a rudimentary map could be made, detailing doorways it had passed through, allowing for rudimentary path planning[48]. A benefit of using the low data-density sensor was that all processing could be done on-board, thereby not relying on reliable communications to a ground station.

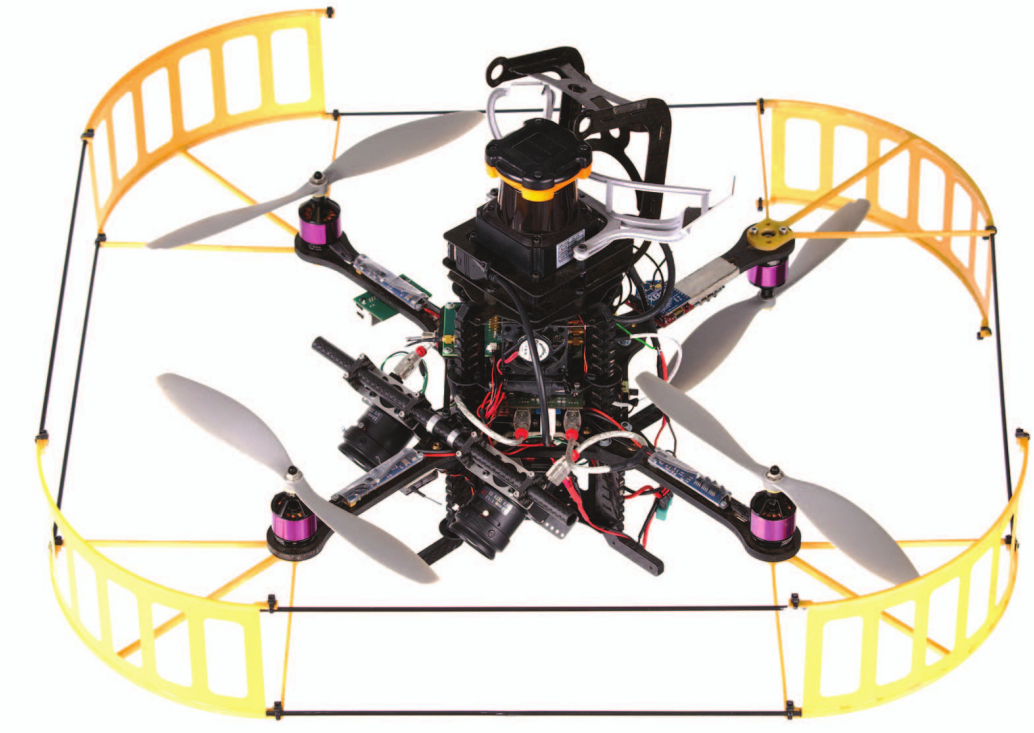


Figure 3.24: The search and rescue UAV[50]

3.5.4 A Search and rescue UAV

Search and Rescue is also another active area in mobile robotics, using robots to help locate survivors and to gather intelligence that the emergency personnel can later use to expedite their efforts. This project looks at developing a UAV that can fly both indoors and outdoors, along with the transition between the two environments and that can autonomously locate objects of interest.

3.5.4.1 Flying Platform

Similarly to that of the MIT-Ascending Technologies UAV in 3.5.2, the UAV utilises an Ascending Technologies Pelican UAV as their base platform (Figure 3.24).

3.5.4.2 Sensors

The UAV uses a LiDAR for detecting walls and movement, an IMU to stabilise itself and four vision cameras. Two of the cameras form a stereoscopic three-dimensional camera pointing ahead of the UAV, along with an upwards facing and a forward facing camera (not shown in Figure 3.24)

3.5.4.3 Localisation and Autonomy

The UAV is fully autonomous, does not rely on reliable communications with a ground station and is claimed to be capable of robust indoor SLAM in unknown and cluttered environments. For localisation the UAV doesn't store a map of the environment, instead it uses known landmarks as it reduces the processing and memory requirements. It, however uses a variant of Iterative Closest Point for odometry between map comparisons to achieve better control [50].

3.6 Summary of Related Projects

There are numerous other similarly related projects than the ones listed above, such as [51, 52, 53, 54, 55, 32, 56]. Although the intended applications vary, most of these UAVs follow a similar setup, using a quad/multi-copter as a development platform, and using localisation algorithms with their positions derived from LiDAR and/or camera data.

The potential robotic control of quadrotors have been demonstrated by the GRASP Laboratory at the University of Pennsylvania and ETH Zürich which show extremely high levels of control. An example being the ability to autonomously fly through gaps slightly larger than the UAV (see figure 3.25a) or precision formation flying (see figure 3.25b). These projects however, rely on external infrared tracking cameras and special fluorescent tags attached to the UAVs to determine their location and

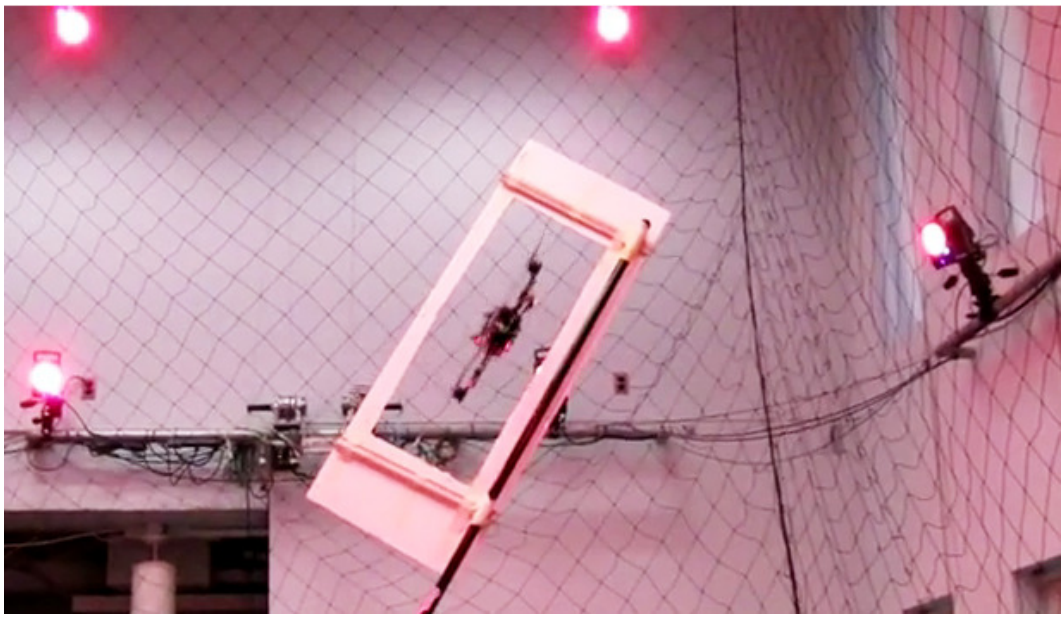
pose. Something which is not viable for this project, but highlights with accurate, robust and rapid access to the location and pose of the UAV, precise control can and has been achieved [57, 58, 59, 60].

The largest academic problem regarding indoor autonomous UAVs appears to be determining the UAVs location and pose without relying on external devices, all while remaining within the payload limits of the platform and achieving robust and precise measurements. SLAM is renowned for requiring large amounts of processing power in-order to operate in real-time. While this is not so much of a problem for *Unmanned Ground Vehicles* (UGV), as they can simply stop and wait between measurements, UAVs cannot simply stop, requiring guaranteed real-time position updates in-order to stay in control.

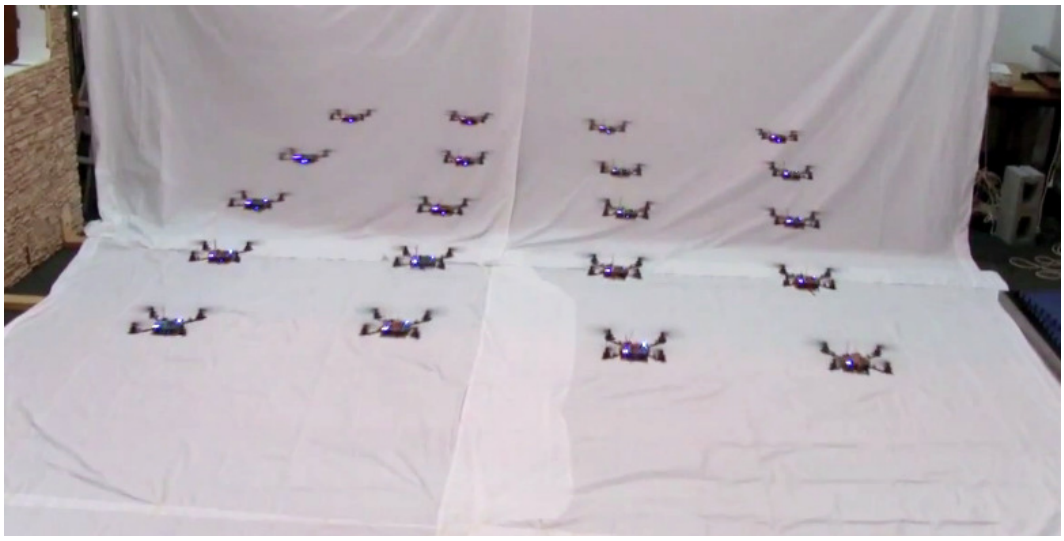
SLAM is a very popular choice for localisation within the academic projects and many advancements have been made, however there is still the fundamental problem that SLAM requires large amounts of memory and processing power. To overcome this issue, research groups have found ways to lessen the processing requirements to make up for the lack in on-board processing power of current computer technology. These include:-

Remote processing: Rather than processing all the data on-board, and being limited by the available on-board processing constraints, several research groups instead relay all the sensor data to a ground-station which processes the data and relays position reports and motion commands back to the UAV. Although this allows for more sophisticated SLAM and planning algorithms to be used it has one significant drawback, namely if two communications are lost at any point then the UAV would be "flying blind".

Assumptions about the geometry of the environment: If assumptions can be made about the operating environment, then the processing requirement can be greatly lowered. If the UAV for-instance is to be flown in a highly structured



(a) A small UAV autonomously flying through a gap slightly larger than itself [61]



(b) Demonstration of swarm control of miniature UAVs[62]

Figure 3.25: Progress of autonomous UAV control demonstrated by GRASP Laboratories

environment (such as an office/corridor style environment), the walls can generally be assumed to be straight and mostly orthogonal. Enabling the SLAM algorithm to match each type of wall as opposed to the raw three-dimensional point-cloud can significantly reduce the processing requirement needed.

Two dimensional instead of three dimensional maps: Although UAVs operate in three-dimensional space, some environments can be approximately represented in two dimensions with the assumption that the UAV maintains a consistent height during the flight and that the environment comprises primarily of featureless vertical walls. Ignoring a dimension drastically reduces the SLAM algorithm's processing and memory requirements, however is not generally suitable for an airborne system. Another approach is to use a 2.5 dimensional model, whereby the two dimensional map is projected into three dimensions. This can offer more robust scan matching, however is unable to accurately represent three-dimensional changes in the environment.

It is not always beneficial to use a "high-tech" solution - keeping the UAV as simple as possible not only reduces development time but also generally lowers the cost of the unit. A notable example of this is the "Sphere Drone" (see figure 3.26) developed by Japanese Ministry of Defence for returning imagery of the insides of buildings, much like the projects above. Instead of using complex algorithms and sensor installations to avoid collisions and provide autonomy, this UAV utilises a large plastic shroud which allows the UAV to be flown into walls. With intelligent design of the CoG the unit also self rights when on the floor and self right if it has rolled across the floor. The UAV is completely tele-operated and is flown manually by an operator. This is achievable as most in-flight collisions during the mission are not a danger to the functionality of integrity of the UAV[63].

Figure 3.26: Sphere Drone developed by the Japanese Ministry of Defence[64]

Chapter 4

Architectural Design

Using the knowledge and information gathered in the *Review of the State of the Art* in chapter 3 and the *Requirements Capture* from chapter 2 the architectural design of the proposed UAV can be created. The architectural design refers to identifying and detailing the sub-systems which will make up the UAV system, ensuring that the sub-systems meet the requirements with minimal development and also function in symbiosis with each other.

4.1 Robotic Platform

Following the earlier discussion in 3.1 regarding established and available flying platforms, the multicopter style design was chosen over the others for the reasons listed in table 4.1. Reliability and performance is paramount to any robotic platform, it was therefore decided that the UAV would be developed using a commercially available multicopter platform. The benefits being that a suitable and capable platform could be acquired at the start of the project, along with the guarantee of reliable controlled and flight electronics "out-of-the-box".

There are numerous multicopter products available, with a mark-able split between "hobbyist" products and more "fully commercial" solutions. "Fully commercial" solutions such as Microdrone shown in figure 4.1a and DraganFlyer shown in figure 4.1b tend to offer products more suitable to aerial surveying and filming. They are certified and have comprehensive end-user support, however all come at a cost

(a) Microdrone MD4-200[65]

(b) Draganflyer X6[66]

Figure 4.1: Examples of "fully commercial" multicopter solutions

of over £10,000. More "hobbyist" style multicopters with slightly lower levels of functionality are available, generally sold as self-assembly kits at a fraction of the cost.

After analysing the different options a HexaKopter designed and produced by HiSystems GmbH was chosen as the development platform as shown in figure 3.5 on page 19. The HexaKopter, with it's six independent rotor and motor controllers, offer a high lifting payload of approximately 2kg and a flight time greater than 10 minutes, longer flight times up to 25 minutes are possible with lighter payloads. An added benefit of using more than four rotors is that there is the possibility of redundancy, where the UAV can remain flying with a non-functioning or broken rotor.

4.2 Autonomy

The reasoning behind automating the UAV was to simplify the operation of flying it, to the extent that an inexperienced operator with minimal training could efficiently use the UAV to complete a set mission. Manually flying a remote controlled helicopter accurately (slightly less so with multicopters) is notoriously difficult and

Classification	Type	Reasoning
Lighter-than-Air	Blimps	Lighter-than-Air designs were disregarded due to the large envelope size required to lift the estimated sensor & battery payload (1.4kg [see 4.4]). Other disadvantages include low manoeuvrability and susceptiveness to wind/drafts, with the benefit of being stable and safer as its inherently buoyant. The estimated size of the envelope, if modelled as a sphere would be in excess of 1.37m diameter or if modelled as a blimp would be 2.58m long and 1m wide, as calculated in Appendix A on page 186.
Fixed Wing	N/A	FW designs were not considered, as it would be both dangerous, difficult and inefficient to pilot an aircraft large enough to lift the payload within the limited space of the interiors of the buildings.
Rotary Wing	Helicopter	A helicopter, especially a contra-rotating style design was considered and met the requirements of slow speed flight and payload capacity (while remaining compact). However, due to their mechanical and dynamic complexity they fall short on the ease of use and maintenance criteria in the specification.
	Multicopter	Much like the helicopter, but without the mechanical complexity. Dynamic complexity (such as rotor balancing) is also reduced as the craft uses more albeit much smaller rotors, thus reducing its effects. Instead of mechanical complexity, multicopters rely heavily on electronics and rate gyros to keep stable, following the approach of configure and forget, which allows for much simpler and robust operation. Another point of note, depending on the number of rotors used, it is possible to achieve redundancy, where the multicopter still is controllable after a motor, motor-controller or rotor failure.

Table 4.1: Comparison of flying vehicles for use in the project

the operator requires substantial training for it to be done safely. For the intended application of this project, the UAV was to be flown indoors, in close proximity to walls, with obstacles and with no line of sight to the operator. To manually fly the UAV the operator would have to rely on on-board cameras to detect and avoid any collision risks, keep the UAV in stable flight and fly the mission objectives. A task which is extremely difficult without contacting any part of the structure, given the reduced situational awareness given by first person view video systems.

The developed system is similar to a fly-by-wire system, where the operator may have the illusion of control, however is actually behind a layer of abstraction. The system monitors the on-board sensor data, allowing it to autonomously maintain stable flight in a set position and avoid potential collisions while attempting to mimic or interpret the control inputs given to it by the operator (see figure 4.2).

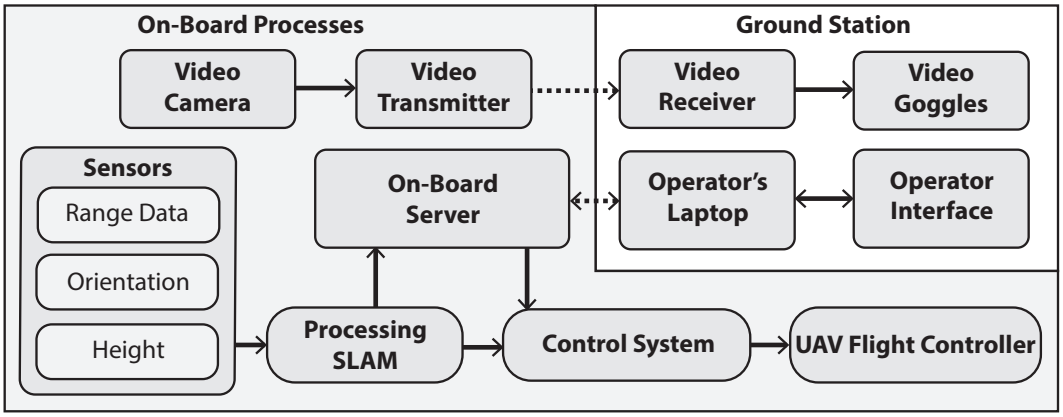


Figure 4.2: A diagram showing the UAV system's high-level architecture

4.3 Localisation - Contribution to Knowledge

To autonomously pilot the UAV and correct for unwanted drift it needs to be aware of its current position and velocity in three-dimensional space. If the UAV were to operate in outdoor environments a system based on INS backed up with GPS to counter for drift would be a robust and easy to implement solution (see 3.3.1). However, as the system is to be used indoors GPS is not a viable option. The

requirements also state that no external devices, including beacons or (off-board) sensors are to be used, limiting the localisation methods to relying solely on the on-board sensors.

Given that the requirements state that the UAV should be capable of creating three-dimensional models of the environment for post-flight analysis, SLAM (see 3.3.2.2) would seem the obvious choice. Using SLAM would require no additional sensors to the ones needed to build the models, minimising both the UAV's complexity, mass and cost.

The most lightweight and cost-effective solution to perform accurate mapping/SLAM on a mobile robot is to use cameras as the primary sensor[67]. Due to the environment the UAV is required to operate in, where lighting conditions may be very poor or completely unlit. The use of visual mapping/SLAM would require powerful on-board lighting systems in order to sufficiently illuminate distant walls (which in the chimney could be as far as 15 metres). Such a lighting system would be heavy and consume a substantial amount of battery power. Also, due to the light source being mounted on-board the UAV, as the UAV moves around inside the structure, the lighting conditions and shadows would vary, making visual SLAM more difficult. Therefore LiDAR based SLAM, although slightly heavier and more expensive, is the better option for this particular application.

SLAM has a few major disadvantages, such as ensuring the robustness of the system and its high processing requirements. As discussed earlier in Section 3.6, there exist a number of similar projects which have chosen to use SLAM for positioning. These projects however have overcome their processing limitations by either relaying their data to a ground-station where more powerful computers can be used, making assumptions about the environment they occupy and/or having access to a prebuilt map of the structure [68].

These methods are not suitable for this project, the geometry of the intended environment will vary between missions and may be used in largely "unknown" environments. Furthermore, due to the design of the structures, which are generally

heavily reinforced, radio shadows may form within the UAVs operating area. If the UAV were to rely on relaying the sensor data for processing, any loss of two way communication would result in the UAV flying "blind" potentially in a runaway state.

No commercial solutions or literature were found detailing a LiDAR based SLAM algorithm or approach which could overcome these limitations. Therefore a large portion of this project was spent on solving the *novel* problem of creating an algorithm which functions in three-dimensions, doesn't rely on pre-programmed features and can function solely on the on-board computer of a computationally constrained UAV, discussed in-depth later in 5.4 on page 89.

4.4 Sensors & Processing

Choosing the correct sensors is a vital component when building a mobile robot and is generally an iterative process, which is difficult to convey in an easy to follow succinct manner. The sensors being used on-board the UAV are listed in Table 4.2 and are influenced by the technologies available (summarised in Section 3.2 on page 22), their successful use in similar projects (see Section 3.5 on page 46) and various manufacturer recommendations.

These sensors were chosen as they offer a minimal and low-weight solution to performing LiDAR based SLAM. The LiDAR (Hokuyo UTM-30LX) was chosen over other manufacturers due to its low-weight, suitable measurement range and scan speed, the UTM-30LX has also been successfully used in a range of UAV and UGVs applications.

An orientation sensor was required to measure the pitch and roll of the UAV and to compensate for the resulting tilting of the LiDAR's planar view. The chosen XSens orientation sensor has been integrated into both civil and military applications with

its key benefits being high-quality pre-calibrated sensor data. The sensor also processes all the raw data internally, outputting only the final calibrated orientation and acceleration values thus lowering the processing demands on the UAV's on-board computer along with development time.

The altimeter (MPX 4115A) is a standard component mounted and integrated into the MikroKopter platform's flight-controller. Testing showed that it offered suitable performance and thus was not changed. The SoNAR (SRF02) digitally calculates the range values on-board the unit, requiring less processing by the UAV's on-board computer and it was chosen based on its performance in the author's previous projects.

At the start of the project the smallest, lightest and most powerful x86 based computer found was the newly released Kontron pITX-SP. More powerful ARM based devices such as smart-phones were emerging, however the x86 architecture was chosen for better driver and application support.

Finally, the Go-Pro camera was chosen as it offered a light-weight solution to obtaining on-board locally recorded High-Definition 1080p footage or higher resolution still images taken at intervals. It also supports simultaneous lower-resolution previewing which allows the same camera to be used for the operator's transmitted video feed.

Device	Mass	Model	Use
LiDAR	290g	Hokuyo UTM-30LX	A high performance, two-dimensional low-weight LiDAR scanner (30metres range 1080points @ 40Hz).
Orientation	65g	XSens MTi	To correct UAV angular pose and detect tilt of LiDAR scanner.
SoNAR	∼5g	Devantech SRF02	For detecting height of UAV when close to the ground (<6 – 7m)
Altimeter	∼5g	Freescale MPX 4115A	Pressure sensor for detecting the height of the UAV when out of SoNAR range.
Camera	190g	GoPro Hero 2	Video feed for operator and acquires images/video for post-mission analysis.
Computer	175g	Kontron pITX-SP	Computer for on-board processing (1.6GHz Intel Atom, 2Gb RAM, 8Gb Storage)

Table 4.2: Sensors and computer used on-board the UAV

Chapter 5

Sub-System Design

The following chapter aims to give a detailed explanation of the developed algorithms, systems and resulting UAV (shown in figure 5.1). Given the limited time available for the project to research, develop and build a solution a number of design philosophies were adhered to throughout the project. Firstly, the emphasis was on designing the UAV using established commercial equipment where possible rather than attempting to re-invent the wheel for the sake of reducing costs. Once a prototype had been built and verified, bespoke equipment could be developed once the exact requirements were known. Secondly, to aid rapid development, testing, debugging and with a view to the potential future commercialisation of the UAV, an emphasis on keeping the whole system and software as simple as possible was maintained throughout.

5.1 System Overview

Given the ease of use and autonomy requirements issued by Sellafield (see Chapter 2), it was decided that the UAV should implement a low-level assisted autonomy control approach (see Table 3.2 on page 45). More specifically, the UAV should autonomously be responsible for stable flight, collision detection and overall safe operation. The operator's job was then shifted towards instructing the UAV where to move to and what specifically to inspect, which enabled an operator with little or no training and experience to successfully and safely carry out inspection missions.

Figure 5.1: The built UAV. Image courtesy of Wired UK [69]

The developed system can be split into four distinct physical sub-systems, namely the UAV platform, sensors, processing and autonomy and the ground station, all of which is discussed in detail in the following sections.

5.2 The UAV

It was decided early in the project that *Commercial Off-The-Shelf* (*COTS*) solutions should be used wherever possible to enable rapid development of a robust prototype. The process of developing and building bespoke UAV and sensor systems is both a difficult and intricate process and requires a substantial development cycle and is out of the scope of this project.

A potential disadvantage of using *COTS* is the loss of flexibility of the device. These devices are usually sold as a "black box", meaning that the exact function of the device may be unknown, only that it should conform with the associated data-sheets. It is often the case that *COTS* devices cannot be modified to suit particular tasks, instead a device needs to be found which best conforms to the intended end application, usually giving rise to compromises.

A significant advantage, however, is that *COTS* devices usually are readily available and are consistent between devices. In the event of damage to a device a new one can rapidly be purchased and fitted, with the expectation that the replacement will function much like its predecessor. This is not always easily achieved using early-stage prototype electronics.

5.2.1 Modifications to the Hexakopter Platform

As mentioned earlier in 4.1 on page 60 a Hexakopter designed and produced by HiSystems GmbH was chosen as the flying base platform. The Hexakopter is a commercially available multicopter platform sold as a kit for primarily hobbyist or

low-budget commercial uses. The kit shown in figure 5.2 on the next page contains all the required electronics, motors and other mechanical components to build the HexaKopter platform.

No significant modifications have been made to the HexaKopter platform or its control systems, these still remain as manufactured. However, a few slight modifications have been made to allow the sensors and devices to be mounted onto the platform, including the addition of two high-power three watt LEDs to help illuminate the environment in low-light conditions, shown in figure 5.3.

5.2.2 On-Board Sensors and Devices

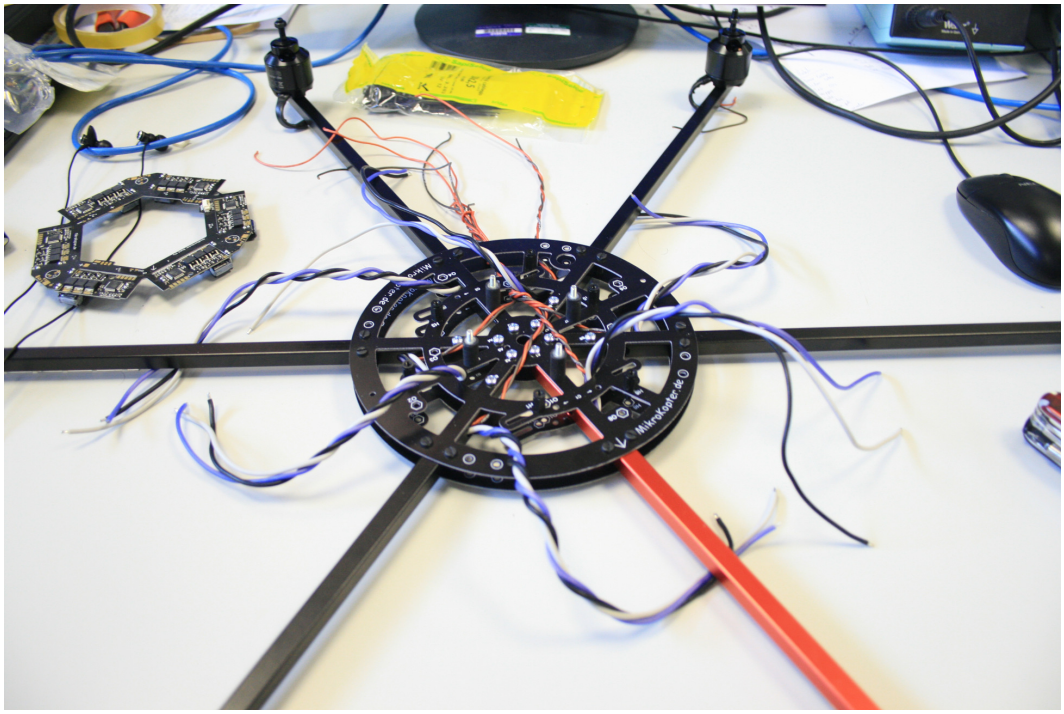
Sensor placement and mounting is a critical aspect of UAV design, ensuring that the sensor has an un-obstructed view while remaining within *Centre of Gravity* (CoG) or weight constraints of the UAV platform. The inherent design of multicopters simplifies this task as the primary mounting point clear from the rotors also happens to be the lateral CoG. This means that sensors and devices can be easily mounted below and above the rotor line, while remaining within CoG limits. Although, for added stability, the heaviest devices should be placed below the rotor-line to increase the pendulum stability of the UAV.

Figure 5.4 shows the mounting positions of the various devices and sensors on the UAV as listed Table 4.2 on page 67. The devices can be split into the upper and lower *device stack* according to their location relative to the vertical rotor-line of the UAV.

5.2.2.1 Upper Device Stack

The upper device stack, shown in figure 5.5 “houses” the Hokuyo *LiDAR* and XSens *IMU* and is mounted directly above the HexaKopter’s flight controller, conveniently offering a similar footprint to the attached devices.

(a) The HexaKopter kit



(b) Partially assembled HexaKopter

Figure 5.2: Building the Hexakopter platform

Figure 5.3: Photo showing the lighting attached to the UAV.
White “headlights” to improve camera lighting.
Red and Blue strip light to improve directional visibility during testing.

Being one of the heavier sensors, the LiDAR should preferably, from a CoG perspective, be placed below the rotor-line. However, if the LiDAR were to be mounted below the rotor-line problems would arise with the landing legs obscuring the scan-line, causing blind spots. Instead the LiDAR was placed at the top of the upper device stack, free from obstructions and potentially slack cables.

The IMU is less particular about its placement, however there are three factors which can degrade performance:-

1. Magnetic interference
2. Vibration
3. Off-axis acceleration

Magnetic interference causes the internal magnetometers to misread leading primarily to erroneous heading data. This can be reduced by distancing the IMU from the motors or other high current devices such as the batteries or motor controllers.

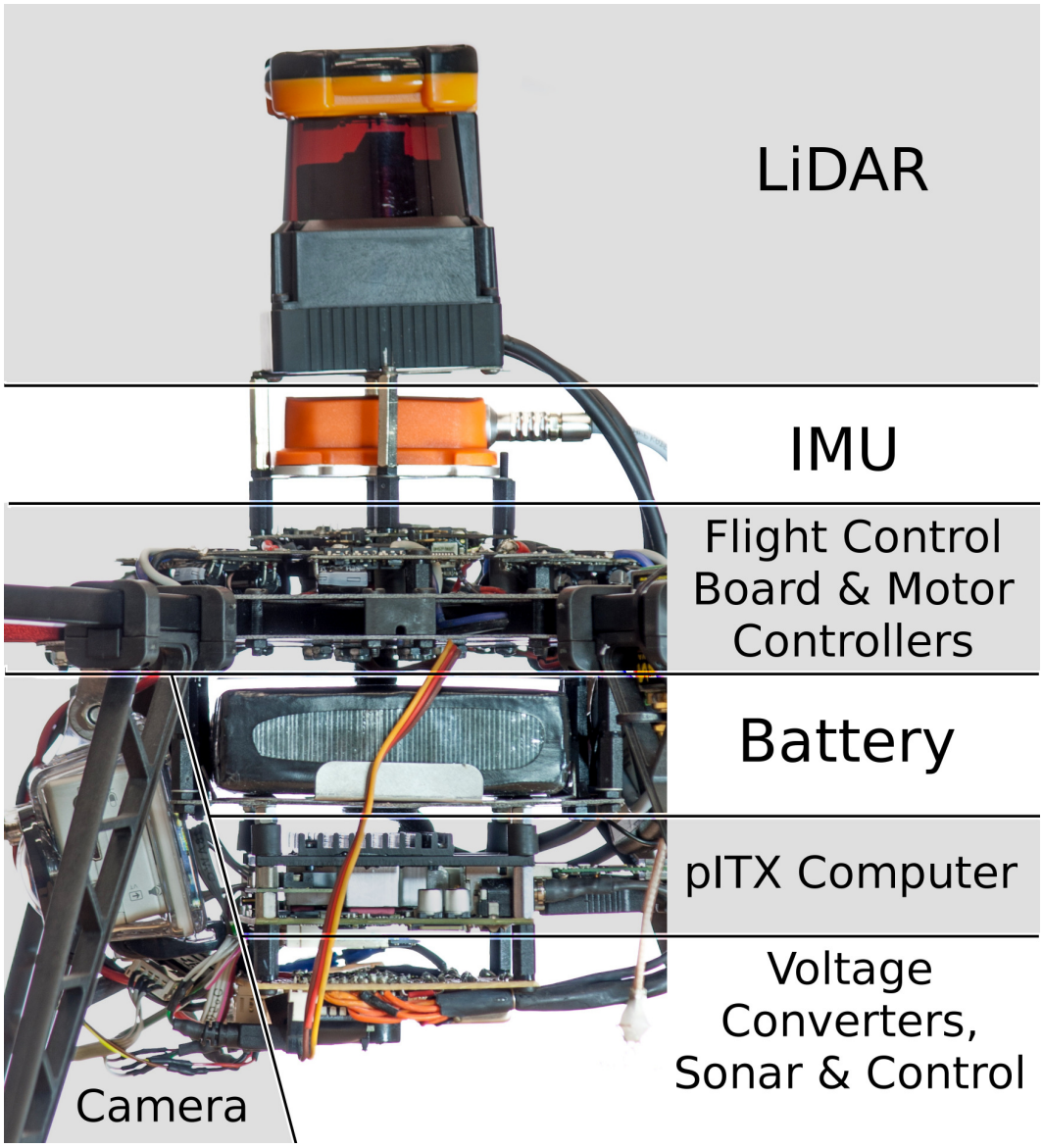


Figure 5.4: A diagram showing the mounting locations of the on-board devices

Heavy vibration will cause degraded performance for both linear and rotary accelerometers and should be dampened whenever possible. Unfortunately powered UAV's, especially rotorcraft, commonly have high levels of vibration from the rotors and motors. To reduce the vibration and possible shock impacts from *less than perfect* landings both the upper and lower device stacks are mounted on rubber dampeners, offering some limited protection.

Off-axis acceleration is caused when the accelerometer is placed away from the centre of rotation while the UAV pitches, rolls or yaws. Due to the lateral offset any rotational movement will induce not only a measured rotational but also a lateral acceleration. To lessen these seemingly erroneous readings, the IMU should be placed as close to the centre of rotation (usually near the CoG) of the UAV as possible.

Taking the above into account it was decided that the IMU should be placed in the upper stack, below the LiDAR. This is the nearest the IMU could be placed to the CoG without mounting it directly beside the motor controllers and high current circuitry, potentially causing disruptive electro-magnetic fields.

5.2.2.2 Lower Device Stack

The lower device stack, shown in figure 5.6, contains the remainder of the on-board devices including a GoPro camera, LiPo battery pack, computer, SoNAR and the *Power Regulation and HexaKopter Interface* board (described in 5.2.2.3).

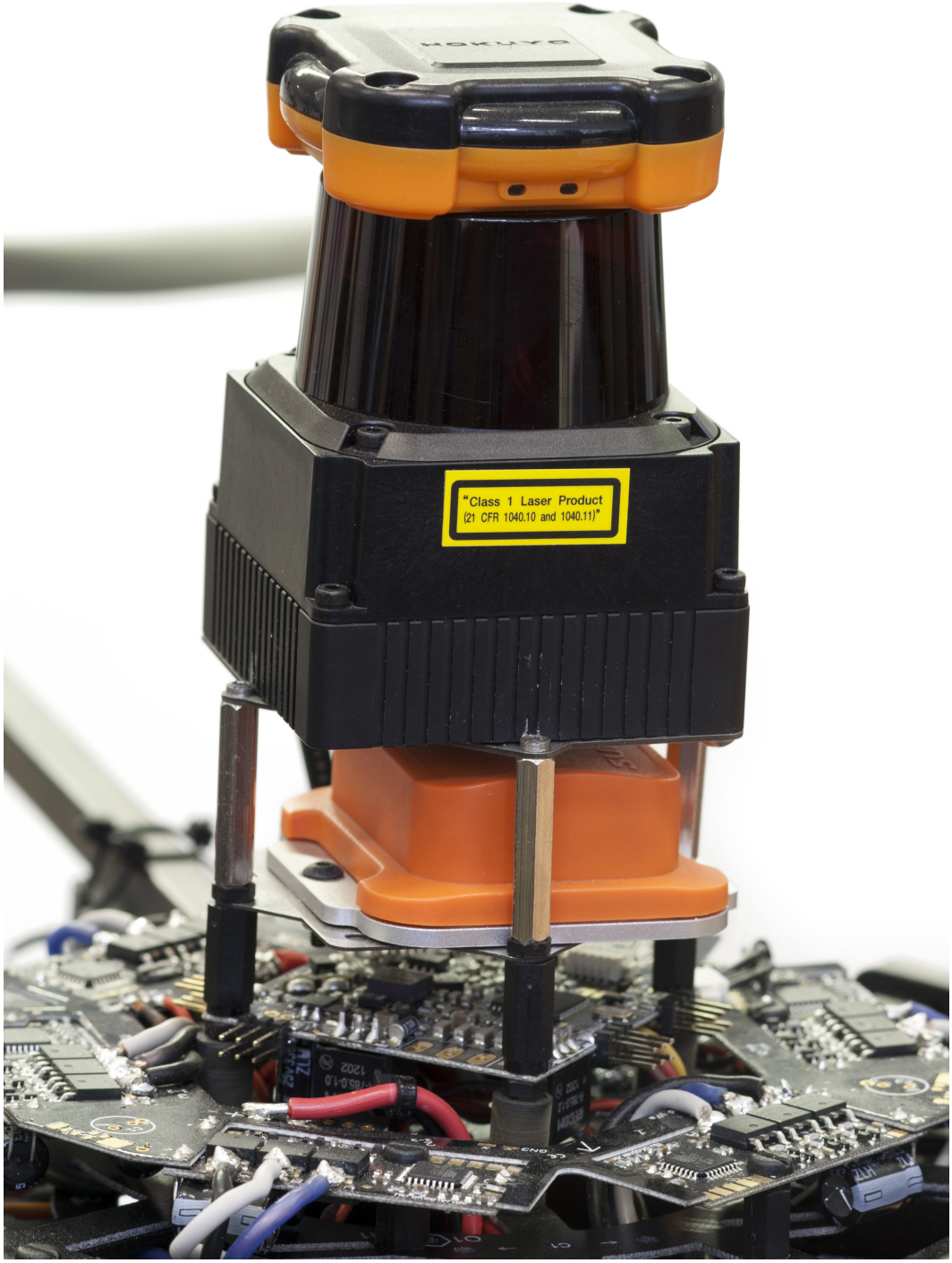


Figure 5.5: The upper device stack on-board the HexaKopter, holding the LiDAR and IMU, mounted above the HexaKopter's flight controller.

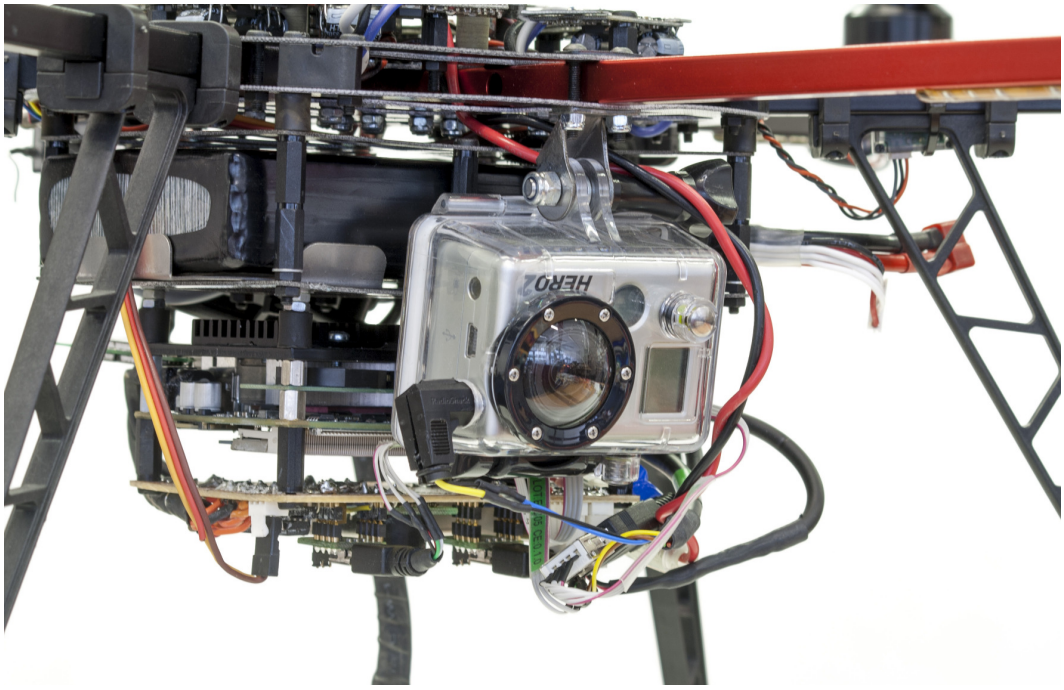


Figure 5.6: The lower device stack on-board the HexaKopter. Holding the battery, computer, power regulation PCB and camera.

5.2.2.3 Power Regulation and HexaKopter Interface

Figure 5.7 shows the only "project custom" electronics used on the UAV. Its purpose is threefold:-

Power Regulation The sensors and devices added to the UAV require a stable electricity supply at a specific voltage. The computer, XSens, ultrasound and receiver circuitry requires 5volt - the LiDAR and video transmitter, however, operates at 12volt meaning that two voltage converters were used, stepping from the LiPo battery's rated 14.8v to the required voltages, while smoothing spikes caused by sudden power draw from the motors.

SoNAR The downwards facing SoNAR (centre of the image) was attached to the board primarily because it is the lowest point of the UAV with an unobstructed

view downwards. The SoNAR was connected to the computer via the leftmost USB to Serial converter in figure 5.7a.

The Hexakopter Interface and Safety Controller This circuit is based on an Atmel 2313 microcontroller which has been programmed to convert commands sent to it from the PC into a Pulse Width Modulated (PWM) style signal, which is then interpreted by the HexaKopters propriatory flight-controller. It also serves a crucial role in safety during testing, as it interfaces with a standard model RC receiver enabling a "safety pilot" to take control if necessary, as described in 5.5.5 on page 113.

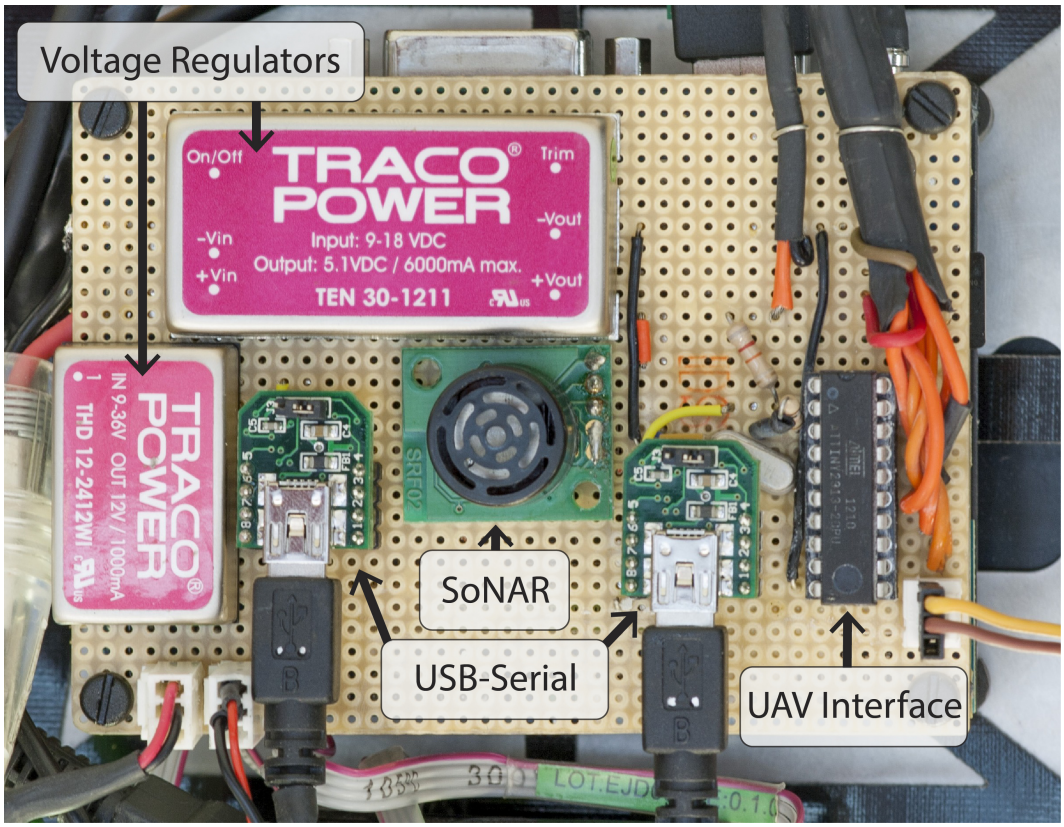
5.2.3 Cost

One of the objectives of the project was to develop the UAV while keeping the unit cost below £10,000. The sensors and systems currently in use on the UAV are listed in table 5.1, and demonstrates that the proposed system design is well within budget.

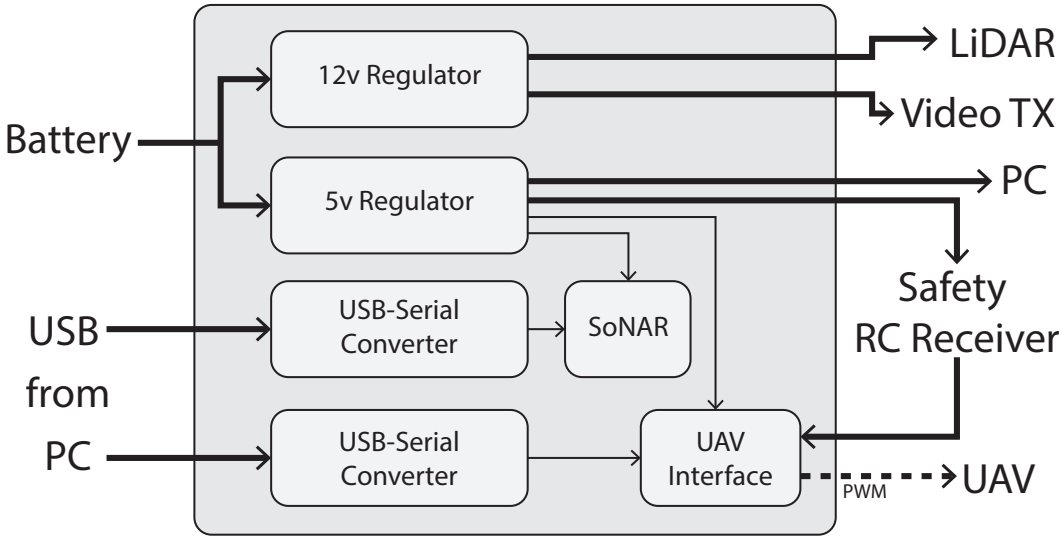
There are a number of items not included in the table, that are, however, vital for the UAV's functions. These include the WiFi router and video receiver. Their cost is not included as they are part of the ground-station, and their reuse can mostly be guaranteed even in the event of a loss of the UAV.

Device	Approximate unit cost at time of order (Ex-VAT)	Year of Purchase
LiDAR	£3223.00	2009
HexaKopter	£1963.00	2012
XSens	£1300.00	2009
GoPro	£268.73	2012
Computer	£257.00	2009
Antennas	£98.49	2010
Powerboard	£76.22	2009
SoNAR	£9.99	2009
Total	∼£7196.43	N/A

Table 5.1: Breakdown of the cost of parts used on-board the UAV



(a) The protoboard design



(b) Flow diagram of the board's functions

Figure 5.7: The custom board holding the power regulators, downwards facing SoNAR and safety circuit.

5.3 On-Board Processing and Control

An abbreviated flow-diagram showing the functions and processes running on the on-board computer is shown in figure 5.8. This section aims to describe the functionality of each of the processes listed within the flow-diagram, except for the sensors and UAV control interface which have already been discussed in 5.2.2.

5.3.1 Localisation

See 5.4 on page 89 for an in-depth explanation of the developed localisation algorithm.

5.3.2 Collision Avoidance

The collision avoidance algorithm presently in use is a simple reactive collision detection algorithm originally designed by the Warwick Mobile Robotics *Search & Rescue* project in 2008[70]. The affectionately named *PieEye* algorithm gathers data from various range sensors, calculates the average distance in each slice, and monitors each slice for any incursion between the average distance and the predetermined severity levels (shown in figure. 5.9).

If a severity level is found to be breached, the algorithm finds the largest detectable open area and notifies the UAV's path planner and control logic of the possible collision risk. As the object gets closer to the UAV further severity levels will be activated resulting in a more aggressive counter-motion being sent to the control system.

At present the first severity level is set at 1.5m from the centre of the UAV. The *PieEye* algorithm directly influences the angular control logic as opposed to the positional control. This allows for basic obstacle avoidance even in the event of the localisation algorithm or associated sensors malfunction. Once the UAV has

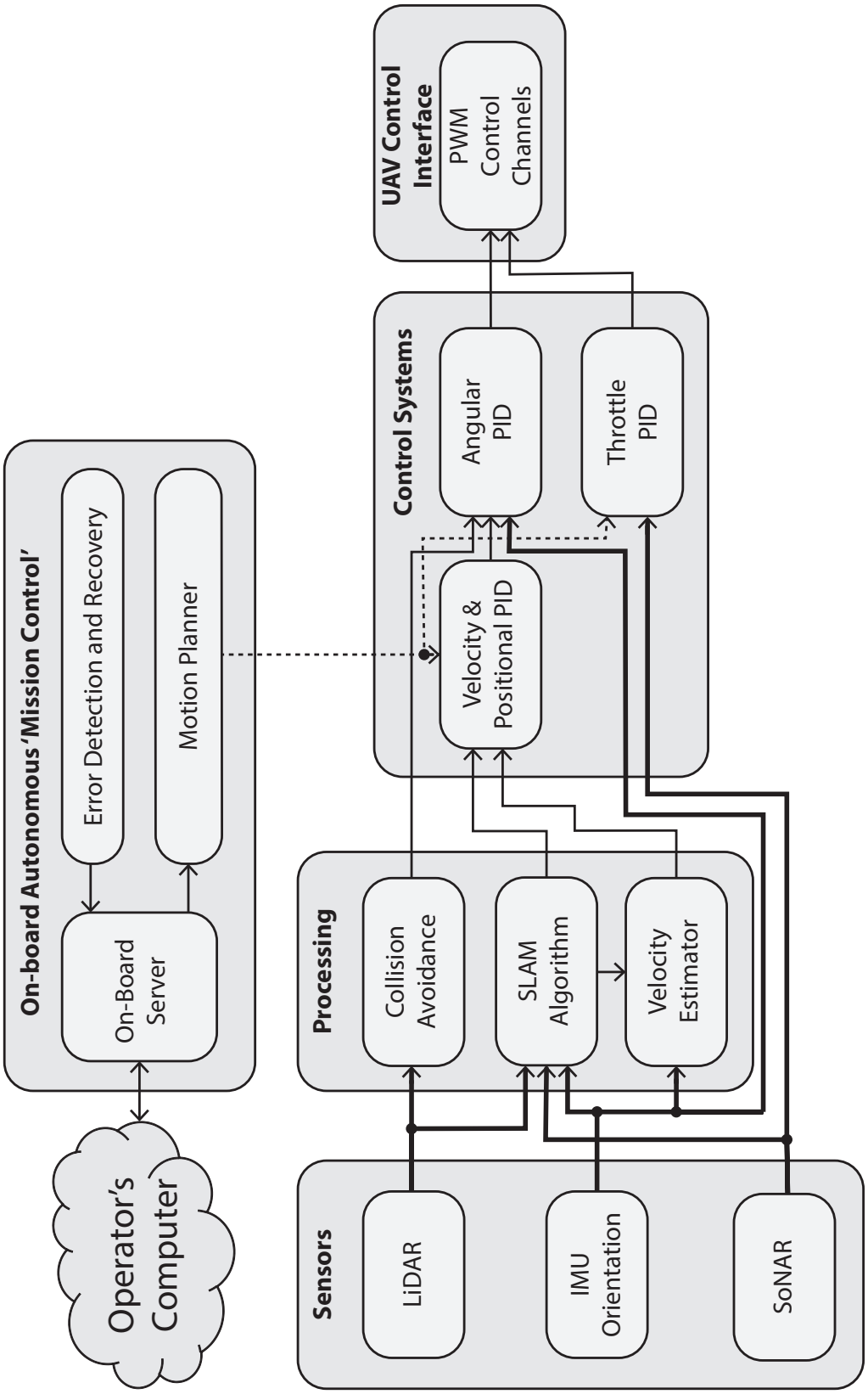


Figure 5.8: An abbreviated flow-diagram showing the processes on-board the UAV

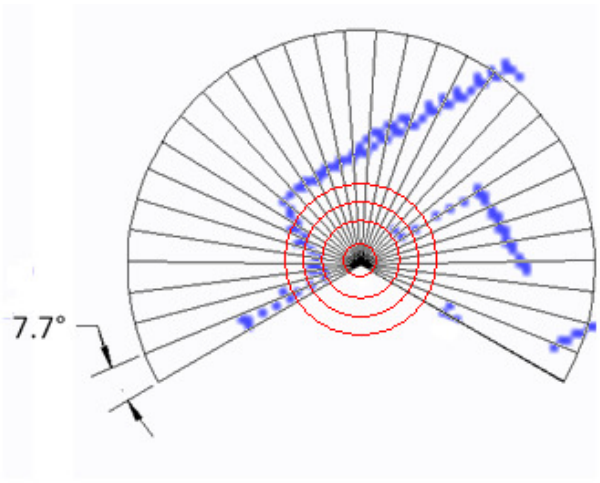


Figure 5.9: Diagram of a 31 slice *PieEye* implementation with five severity levels[70]

cleared the obstacle and is no longer considered a threat, control is returned to the positional control logic and a new position waypoint is set to the current position to minimise the chance of the UAV trying to the return to the position of the collision.

5.3.3 Control Theory

Knowing the location of the UAV and knowing where it should be going is only the start of the larger challenge of actually flying the UAV. Flying machines generally have many non-linear dynamics acting on them and the control inputs can have variable effectiveness depending on a large number of factors. This is one of the reasons why helicopters are particularly difficult to fly.

One of the primary motives for purchasing a commercial UAV platform was that the dynamics to some extent were solved internally on the UAV's flight controller, meaning that any commands sent to the UAV would result in a positive and commanded movement. This would allow the UAV's flight controller to monitor the UAV's pose and subsequently vary the individual motor speeds to keep the commanded orientation as best as possible without additional sensors or algorithms (axes defined in Figure 5.10). The on-board computer and its algorithms could then concentrate

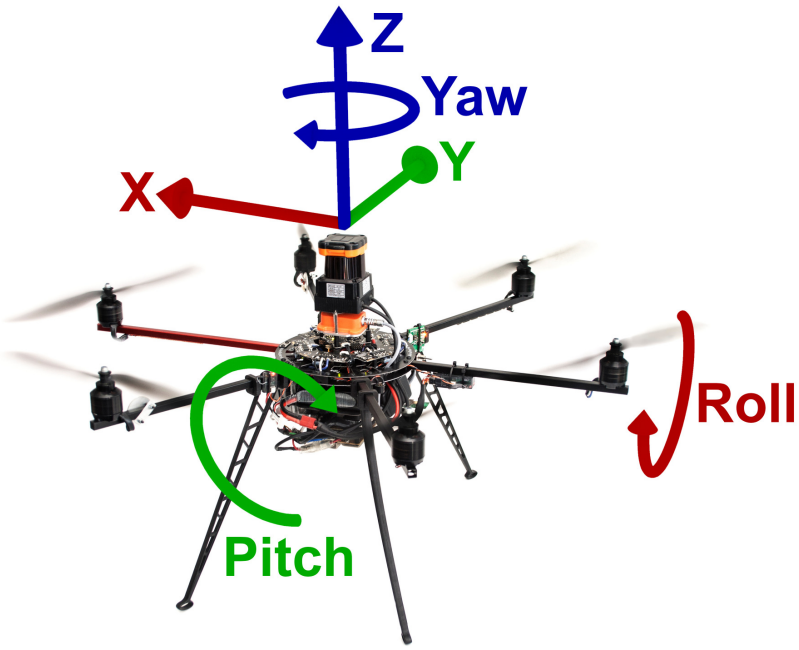


Figure 5.10: Definition of axes on the UAV

on the "low-speed" dynamics, such as hovering in a set location or transversing the environment at a set velocity.

There are a number of projects, which are researching the challenge of solving the dynamics of an indoor UAV platform, as briefly discussed in 3.6 on page 55 and it has to a large extent been solved. Implementing their designs, however, takes considerable knowledge and skill within the control theory field and is out of the scope of this research project, which is primarily focusing on investigating and developing a suitable method for on-board indoor localisation.

For this work, a non-model based control system was used as the system stability and dynamics have yet to be characterised and modelled, which limits the use of more advanced control methods, such as "Feed Forward" techniques. Instead the UAV is *temporarily* relying on simple *Proportional Integral Derivative* (PID) control until a more intelligent control strategy can be fully implemented. Simple PID control has been proven to function on similar UAV setups [12].

The UAV utilises three separate PID controllers to control the UAV, namely - positional, angular and height, as discussed in the following three sub-sections. Given the lack of a mathematical model of the built UAV, and that creating and verifying one is very time consuming, made simulating the UAV impractical. Instead the PID gains were iteratively tuned through real-world flight testing. By logging the individual P, I and D terms during short flights it was possible to selectively tune the gain values resulting in stable, although *not* optimised, PID control.

5.3.3.1 "Height" PID loop

The "height" PID loop controls the throttle, which primarily affects the vertical speed of the UAV. It is the simplest PID controller in that it only takes its information from a single source, the SoNAR sensor, and the set-point from the motion-planner in the server application. To prevent spurious throttle changes the SoNAR data is filtered and the output of the PID loop is capped to a certain range of values.

5.3.3.2 "Angular" PID loop

Although the HexaKopter's internal flight controller handles the fast dynamics of the UAV, the interface between the PC and the HexaKopter platform is analogue and arbitrary. This means that an angular position can easily be commanded, but the actual angle achieved can vary due to many environmental and internal influences.

For this reason an "angular" PID loop is used, which takes the high accuracy roll, pitch and heading data from the XSens IMU and enables the UAV to be flown accurately and consistently to a set angular position.

5.3.3.3 "Positional" PID loop

The "positional" PID loop takes data from the localisation algorithm and the velocity estimator (fixed-frame co-ordinates from the take-off position and orientation)

and outputs a target angle for the UAV, fed into the "angular" PID.

The primary component of the "positional" PID is a PID loop centred on controlling the UAV's velocity, the reason being the compatibility between the way in which a PID loop operates and the way in which a rotary wing craft flies. An ideal rotary wing craft, when level, will not accelerate and will maintain its current velocity, only varying when a bank/pitch angle is applied. In order to fly to a position the UAV first needs to be accelerated towards the target setpoint, and then needs to decelerate to stop perfectly at the desired point.

If a solely positional PID is used, then the UAV will always overshoot the target setpoint. This is due to the fact, that the PID's output will be "zero'ed" around the setpoint, meaning, if the output is the angular orientation, that the UAV will only start decelerating once it has overshot the target, due to the non-linearity between position and requested angle.

The design of the "positional" PID loop through the testing phases of the project is a fusion of a PID and a PI loop, PID for velocity and a PI loop for positioning. Due to position being the integral of velocity the control system can be visualised as a $P(I + I^2)D$ loop shown in full in eqn. 5.1.

$$u(t) = K_p[Vel(\tau)] + K_d[Acc(\tau)] + K_i[Pos(\tau)] + K_{ii}\int_0^{\tau}[Pos(\tau)]d\tau \tag{5.1}$$

Where:-
K_p, K_d, K_i and K_{ii} are the gain constants and $u(t)$ is the output of the PID as the UAV's desired angle in either roll or pitch.
Acc is the *error* between the acceleration measured by the IMU and requested by the motion planner
Vel is the *error* between the velocity calculated by the velocity estimator (see 5.3.4) and the requested velocity by the motion planner.
Pos is the *error* between the blended position from the velocity estimator (see 5.3.4)

and the localisation algorithm (see 5.4), and the position requested by the motion planner.

5.3.4 Position and Velocity Estimator

The localisation algorithm is not ideal and will have errors in its position measurements, which, when differentiated to calculate the current velocity, can introduce significant high-frequency noise, shown in 6.3 on page 123. A common solution to this problem is to perform data fusion between the IMU and the localisation algorithm. As discussed earlier in the feasibility section regarding INS (3.3.2.1 on page 40), inertial navigation suffers from drift caused by, amongst other things, bias in the accelerometers. Due to the acceleration being double integrated when calculating position, large errors can accumulate quickly.

This drift can be minimal if the absolute positions calculated by the localisation algorithm are used to correct the integral error. The benefit of this is two-fold, the control algorithms benefit from getting positional and velocity updates at the rate of the much faster IMU as opposed to the slower localisation algorithm, as PID control requires that the sampling rate and system dynamics meet the Nyquist frequency criteria in-order to function correctly (where the sample rate is at least double of the signal being controlled to prevent aliasing[71]) . Additionally, there is a delay when collecting the LiDAR data, processing and determining the location, a characteristic that can be significantly reduced by allowing the IMU to estimate the current position. Meanwhile the localisation algorithm is used to correct drift on stored past data.

5.3.5 Motion Planner

The role of the motion planner is to aid the control logic interpolating all position instructions to avoid step inputs that may introduce instability into the PID controllers (5.3.3). To provide additional stability and to minimise overshoot in the

PID controllers the velocity of the interpolated positions are ramped both up and down using predefined constants.

The motion planner is pre-programmed with a maximum velocity, ramp-up and ramp-down accelerations. When a new position is requested the motion planner uses the equations given below.

Given the constants:-

- Maximum velocity:$Vmax = 0.5m/s$
- Ramp-up time:$T_{ru}\left(\uparrow_{0m/s}^{Vmax}\right) = 5seconds$
- Ramp-down time: $T_{rd}\left(\downarrow_{Vmax}^{0m/s}\right) = 5seconds$
- Ramp-up acceleration: $A_{ru} = \frac{V_{max}}{T_{ru}}$
- Ramp-down acceleration: $A_{rd} = \frac{V_{max}}{T_{rd}}$

The variables:-

- Distance to final position:D_{xy}
- Current velocity of the UAV: U_{xy}
- t_{ru}, t_{cruise} and t_{rd} are the calculated respective ramp-up, cruise and ramp-down times needed to complete the motion.
- D_{ru}, D_{cruise} and D_{rd} are the calculated respective ramp-up, cruise and ramp-down distances needed to complete the motion.

For a *large* D_{xy} where $Vmax$ can be reached the timings are calculated by:-

$$t_{ru} = \frac{U_{xy} - Vmax}{A_{ru}} \tag{5.2}$$

$$t_{rd} = T_{rd} \tag{5.3}$$

$$D_{ru} = U_{xy} t_{ru} + 0.5 A_{ru} t_{ru}^2 \tag{5.4}$$

$$D_{rd} = U_{xy} T_{rd} - 0.5 A_{rd} T_{rd}^2 \tag{5.5}$$

$$D_{xy} = D_{ru} + D_{cruise} + D_{rd} \tag{5.6}$$

Therefore to calculate t_{cruise}:

$$t_{cruise} = \frac{D_{xy} - D_{ru} - D_{rd}}{V_{max}} \tag{5.7}$$

Thus the ramped and interpolated position can be given as:

$$D_{target} = \begin{cases} U_{xy} t + 0.5 A_{ru} t^2 & t < t_{ru} \\ D_{ru} + V_{max} \left(t - t_{ru} \right) & t_{ru} < t < \overleftrightarrow{t_{cruise}} \\ D_{ru} + D_{cruise} + U_{xy} \left(t - \overleftrightarrow{t_{cruise}} \right) - 0.5 A_{rd} \left(t - \overleftrightarrow{t_{cruise}} \right)^2 & \overleftrightarrow{t_{cruise}} < t < t_{total} \\ D_{xy} & Otherwise \end{cases} \tag{5.8}$$

Where: $\overleftrightarrow{t_{cruise}} = t_{ru} + t_{cruise}$ & $t_{total} = t_{ru} + t_{cruise} + t_{rd}$

For *small* motions it may not be possible to reach $Vmax$ given the ramp-up and ramp-down constraints. In this case the motion planner ignores the cruise stage and calculates a partial ramp-up and ramp-down scenario. As $Vmax$ cannot be reached Eqn. 5.2 gains an unknown and the assumption of Eqn. 5.3 is no longer valid. A different method of obtaining t_{ru} and t_{rd} needs to be used. Given the assumption $t_{cruise} = 0 \therefore D_{cruise} = 0$ and that the ramp-up and ramp-down time constants are equal such that $T_{rd} = T_{ru} \therefore A_{ramp} = A_{rd} = A_{ru}$, Eqn. 5.4 – 5.6 can be rearranged to:-

$$t_{ru} = \frac{\sqrt{2}\sqrt{A_{ramp}^2 (2 A_{ramp} D_{xy} + U_{xy}^2)} - 2 A_{ramp} U_{xy}}{2 A_{ramp}^2} \tag{5.9}$$

$$t_{rd} = \frac{U_{xy} + A_{ramp} t_{ru}}{A_{ramp}} \tag{5.10}$$

5.3.6 Error Handling

Error handling is an important aspect in all software, especially mission critical software such as the software on-board the Hexakopter. The error detection capabilities of the Hexakopter can be split into two categories, passive and active. A "passive" error is captured by the individual classes in the code, where an error report is submitted to the error handler and the class tries its best to solve the problem internally. An "active" error is, for instance, where a sensor or mission critical task simply stops responding. After a specified elapsed time the error handler will re-initiate the thread or sensor and notify dependant classes. This is detected through the use of an extensive watchdog system, where each critical class has to report to the error handler as part of its normal operation to prove that it is functioning correctly.

The unfortunate scenario on the Hexakopter is that at present, due to weight and size restrictions, there are no redundant sensors (except height to a degree). When a sensor drop-out is detected, depending on the sensor or thread that has stopped working, the error handler has pre-programmed responses and failure modes to try and best recover control, as described in Table 5.2. Notifications of all detected errors are sent to the operator to aid with situational awareness, which is further explained in Section 5.5.1.

5.4 On-board Localisation

In continuation of the systems outline in 4.3 of the *Architectural Design* chapter, this section describes, in-detail, the developed localisation algorithm, its known limitations and strategies put in place to avoid them.

There are many existing *base algorithms* which can be used to enable the robot to perform SLAM, as discussed in 3.3.2.2 on page 41 of the *Review of the State of the Art* chapter. *Base algorithms* are tried and tested algorithms, however, they are

Type of error	Immediate response	Secondary action
Ultrasound Timeout	Set throttle to slightly below raw hover to prevent runaway and establishes a controlled descent.	Clear port connections and attempt to reinitialise sensor device class then notify operator
IMU Timeout	Set raw hover pitch, roll and yaw trim (should level Hexakopter to avoid further lateral acceleration)	Clear port connections and attempt to reinitialise sensor device class then notify operator
LiDAR Timeout	Clear port connections and attempt to re-initialise sensor device class	Notify operator
Localisation Error	Switch to INS localisation only.	If mapping is not restored within a few seconds, the old map is discarded and a new map is created.

Table 5.2: Error recovery responses for critical processes

often built for different projects or environments, thus not optimised nor tested for the required new task. In line with the UAV's requirements that it should be able to operate in an unknown environment without re-configuration or learning, a base algorithm was chosen, which doesn't rely on landmark/feature detection. One such algorithm is the popular *Iterative Closest Point* (ICP) algorithm[39, 40, 41, 72].

5.4.1 The Iterative Closest Point Algorithm

The ICP algorithm, although not the most modern, has a few significant advantages towards the application of the project. Firstly, the algorithm functions on unprocessed point-cloud data, meaning that the raw LiDAR sensor data can be used without the need for further processing. Secondly its simplicity allows for rapid implementation and modification compared to more complex probabilistic methods generally developed and maintained by large research groups.

The ICP algorithm was initially developed to *register* three-dimensional point clouds. *Registration* being the process of determining the transformation and rotation offset between two point-cloud datasets, determining their relation in space.

5.4.1.1 Principles of Operation

ICP is, as the name suggests, an iterative algorithm requiring many passes to fully *register* or "align" the two datasets. Unlike some probabilistic registration algorithms ICP requires the two point-clouds to be in the approximately correct orientation and location allowing the algorithm to fine-tune it's position as opposed to seeking and matching from an expansive dataset.

Algorithm 5.11 shows an implementation of the ICP algorithm, which can be split into the following steps[73, 74]:-

1. Firstly, the two point-clouds are compared and the nearest neighbour of each of the scan/model points (m_i) is found in the map dataset (d_i) and the respective mean distances c_m,c_d are calculated using:.

$$c_m = \frac{1}{N}\sum_{i=1}^{N} m_i \qquad c_d = \frac{1}{N}\sum_{i=1}^{N} d_i \tag{5.11}$$

2. To converge the two point-clouds both the translation and rotation needs to be estimated. There exists a number of ways to calculate the rotation matrix ($\mathbf{R}$), such as through *singular value decomposition* (SVD), orthogonal matrices and unit quaternions[74]. When solving the rotation through SVD the rotation can be calculated using $\mathbf{R} = VU^T$ where $V$and U are derived from the SVD $H = U\Lambda V^T$, where H is a correlation matrix calculated from:

$$H = \sum_{i=1}^{N} m'^T_i d'_i \tag{5.12}$$

   where $m'_i = m_i - c_m$ and $d'_i = d_i - c_d$

3. Given the rotation matrix ($\mathbf{R}$) calculated above the translation ($\mathbf{t}$) can then be found through a least squares solution using $\mathbf{t} = c_m - \mathbf{R}c_d$.

$$E(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{N_m}\sum_{j=1}^{N_d} \omega_{i,j} \left\| m_i - (R\hat{d}_j + \hat{t}) \right\|^2 \tag{5.13}$$

4. The error function is then calculated, equation 5.13 can be simplified to:

$$E(\mathbf{R}, \mathbf{t}) = \frac{1}{N} \sum_{i=1}^{N} \|m_i - (Rd_i + t)\|^2 \tag{5.14}$$

given that:

$$N = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} sgn\, \omega_{i,j} \tag{5.15}$$

Through comparing the error function with the previous iteration $\|E_{i-1}(\mathbf{R}, \mathbf{t}) - E_i(\mathbf{R}, \mathbf{t})\|$ the improvement of the applied transformation can be gauged. If the error falls below a threshold or if the maximum amount of iterations are reached then the iterative loop is broken and the two datasets are considered aligned, otherwise another iterative step is performed (goto 1).

5.4.1.2 Limitations

Fundamentally there are a number or limitations to the ICP algorithm, which can cause erroneous point-cloud registrations. Although the basic algorithm given above has been proven to minimise the error function, it cannot detect whether the calculated solution is a local or global minima. Errors in the point pairing process is also one of the reasons why the ICP algorithm is required to be iterative, rarely giving a correct "one-shot" solution. Figure 5.12 shows the inherent difference between true point-pairing (Figure 5.12a), and a nearest neighbour search (Fig 5.12b).

5.4.2 Nearest Neighbour Searches

The most computationally intensive step of the ICP algorithm is determining the *Nearest Neighbour* (NN) of each of the points in the dataset. In order to get the algorithm to function on a computationally contained UAV the nearest neighbour search, a "brute force" approach to finding the points, cannot be used. Instead optimised spatial sorting methods need to be used, such as Octree and Kd-Tree algorithms.

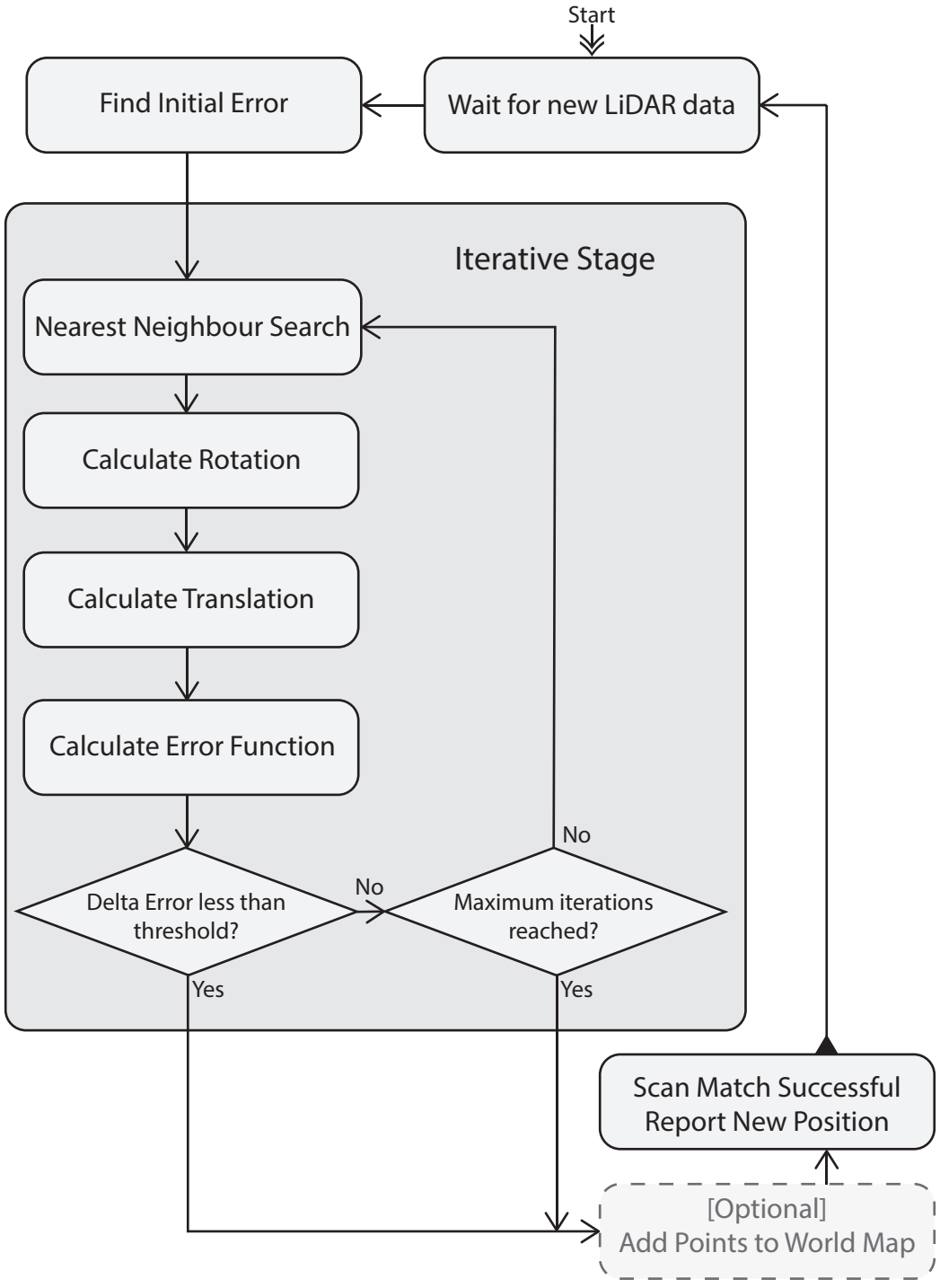
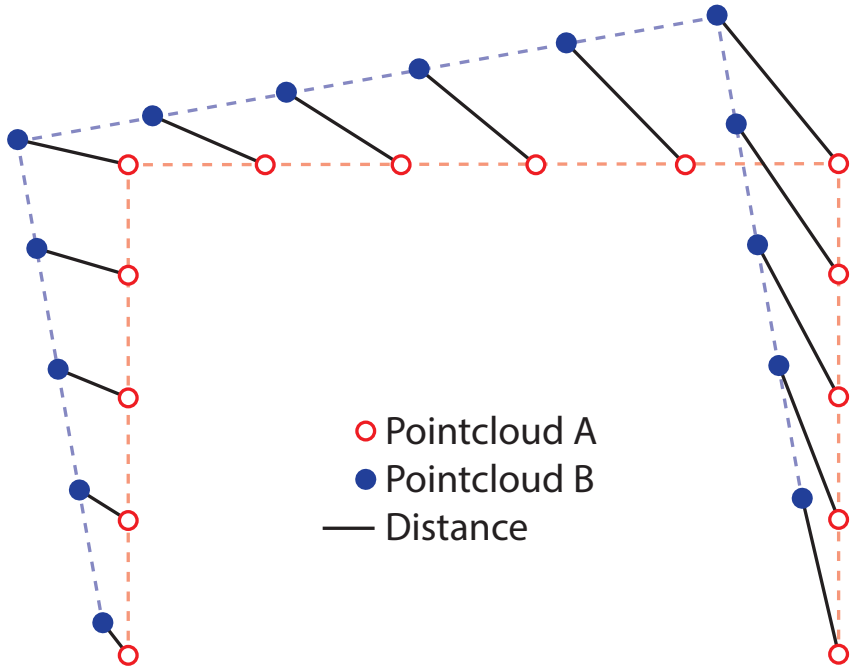
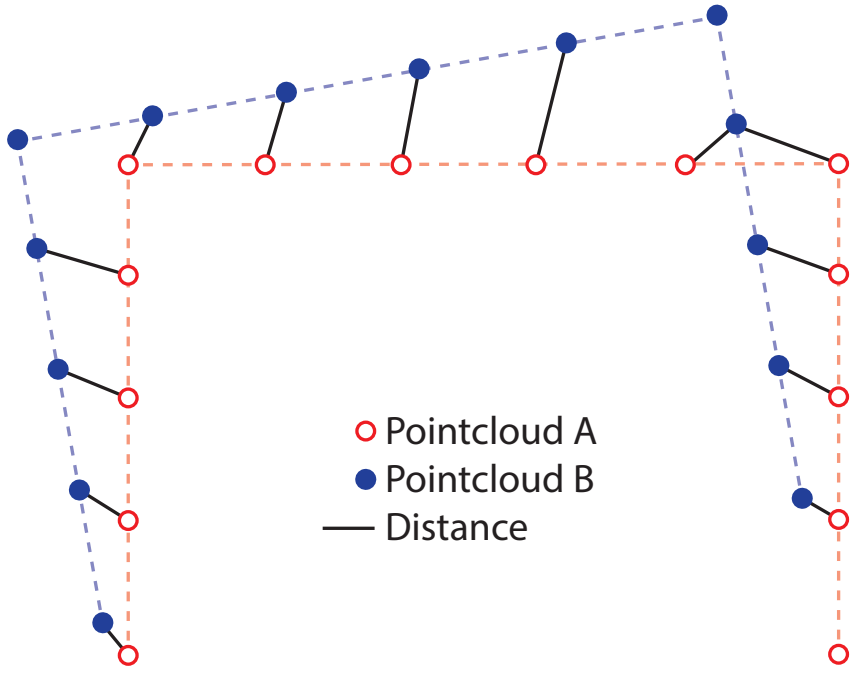


Figure 5.11: Flow diagram of the ICP algorithm as described by[74]



(a) Ideal, correctly assigned point-pairs



(b) Typical output from a nearest neighbour search

Figure 5.12: Errors introduced during point-pairing/nearest neighbour searches

5.4.2.1 Brute Force Search

A *brute force* search or *exhaustive* search is a sorting algorithm, which searches every possible combination before returning an answer. The advantages are, that the data does not require prior sorting or tagging and is easy to implement. However, the search is very inefficient, as each search requires that each datapoint in the whole dataset is inspected for every point in the model dataset, thus rapidly becoming unwieldy for larger datasets.

5.4.2.2 Octree

The *octree* approach is a significant improvement over the *brute force search*, in essence providing a structure to the dataset allowing for rapid exclusion of data-points. As shown in Figure 5.13 the dataset is stored in a hierarchical tree structure, where each node has eight children, which are spatially distributed as an evenly subdivided cube. The nodes at the lowest layer of the tree hold their respective data-points located within their bounds and a fixed number of layers are used, generally chosen to optimally represent the intended dataset [75].

Data is retrieved through recursively checking which of the eight nodes the requested co-ordinates exist in, until the lowest layer is reached and nearest node with data-points are found. Depending on the number of layers used and thus the grid size of the lowest layer, it may contain multiple data-points. These data-points, which are a small subset of the original data set, can then be realistically searched using, for-instance, the brute force algorithm.

When the new points are added to the dataset each point has to transverse the tree to find their respective low-level layer node, which for each additional layer adds further processing overheads, but results in a smaller “bin” for subsequent brute force searches.

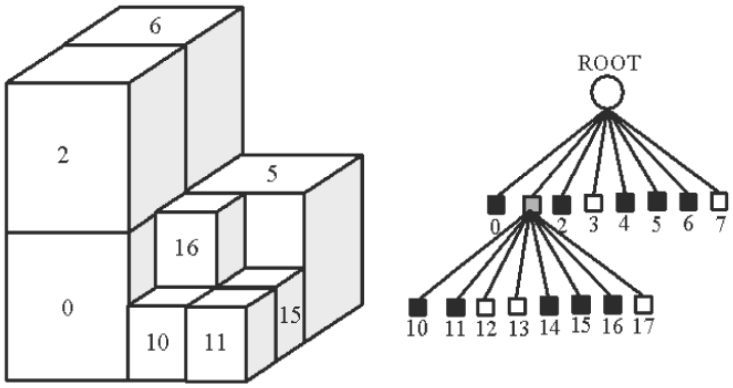


Figure 5.13: Subdivision structure of an Octree[76]

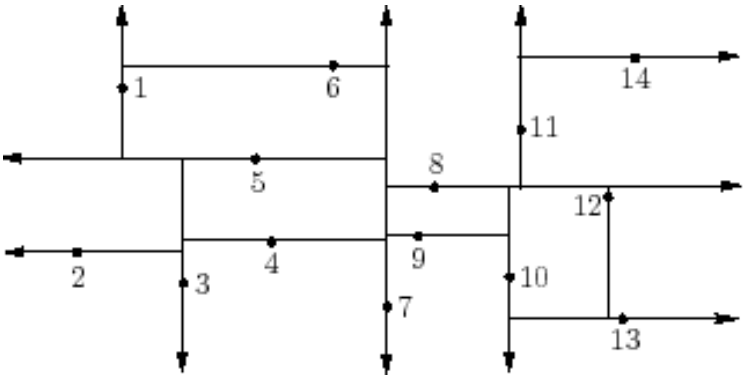
5.4.2.3 Kd-Tree

Kd-Trees, much like the Octree, utilises a binary tree structure to manage the dataset. However, unlike the Octree, where each node is a section of the parent's spatial bounds, the Kd-Tree operates on the principle that the individual points in the dataset determine the subdivisions, and all points are assigned an individual node on the tree structure.

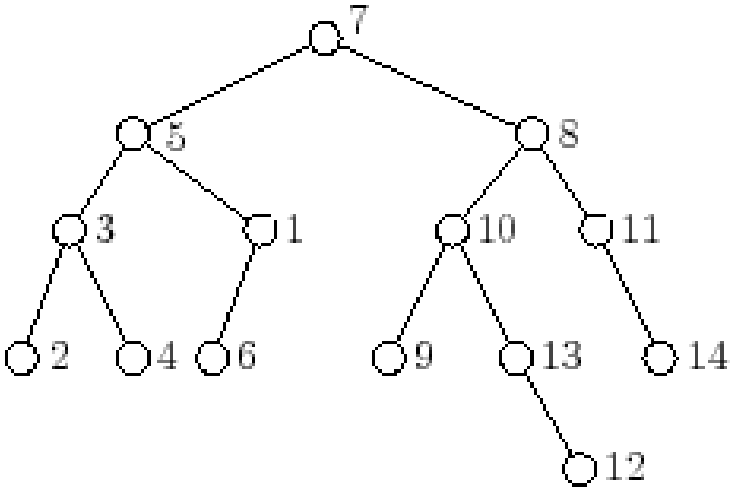
Figure 5.14a shows the segmentation of a simple two-dimensional dataset with the resulting tree structure shown in Figure 5.14b. Firstly, an initial point is chosen, an intersection is made in the x-axis forming the first node and the second node is split using a greater-than/less-than criteria given its relative x-axis location of the intersection. For the second intersection a different axis is used, in this case the y-axis. This method is repeated until all points are assigned nodes.

5.4.2.4 Approximate Nearest Neighbour Search

A significant performance gain can be achieved if the application allows the use of an *Approximate Nearest Neighbour* (ANN) as opposed to the absolute/true nearest neighbour. Searching for an approximate nearest neighbour allows the search algorithm to utilise optimisation algorithms to rapidly guess the location of a nearby



(a) Two-dimensional representation of the structure of the tree[77]



(b) The resulting tree structure[77]

Figure 5.14: Structure of a basic two dimensional Kd-Tree

point, which may not always be the absolute closest neighbour. The tolerance of this can be set in the algorithms parameters.

A popular ANN solution is the *Fast Library for Approximate Nearest Neighbour* (FLANN)[78]. FLANN is a library composed of a collection of nearest neighbour search algorithms, developed to be easily implemented into existing software. It also automatically structures the data into an optimised *Kd-Tree*, requiring only to be sent the initial unstructured dataset and any subsequent new points for it to function.

5.4.3 The Developed SLAM Algorithm

During the course of the project the on-board SLAM algorithm has undergone many iterations, with the objective of achieving accurate, robust and rapid real-time localisation of the robot within an unknown, unstructured and potentially cluttered environment on-board a computationally constrained UAV. The algorithm is based on the ICP method for *scan-matching* (the process of matching a LiDAR scan with a *world* point-cloud), utilising FLANN for fast nearest neighbour searches along with a number of application specific optimisations, all of which are further explained in 5.4.3.1.

The developed SLAM algorithm is *novel* and *unique*, in that no other UAV found in literature is capable of real-time, on-board, fully three-dimensional, LiDAR based SLAM, which can operate in an unknown and unstructured environment. Similar combinations of ICP and approximate nearest neighbour based algorithms have been successfully deployed on UGVs to explore and map unknown environments[79, 80], however, this has not been transferred and solved for an indoor UAV solution as proposed herein.

5.4.3.1 Principles of Operation

As shown in the analysis of the SLAM algorithm in section 6.4 on page 125 of the Sub-System Testing chapter, the three most processor intensive tasks of the basic ICP algorithm are primarily the nearest neighbour search, calculating the error function followed by the function calculating the rotational offset.

Due to the limited amount of processing power available on the UAV the algorithm has been modified to minimise and optimise these tasks as far as possible. These are the *four* major modifications and optimisations, which have been applied to the basic ICP SLAM algorithm:-

1. The use of an approximate nearest neighbour search, FLANN, to lower the processing requirements and hence speed-up each iteration.
2. Reduction of the number of points used from each LiDAR scan. Fewer points require less processing and result in a less dense world map which requires less memory.
3. Removal of the error function from the iterative part of the algorithm, instead relying more on a before and after scan-matching error metric.
4. Removal of the function which calculates the best rotational fit. Instead relying solely on sensor data to detect rotation, only using the SLAM algorithm to effectively calculate lateral translation. This also helps to reduce the time taken for each iteration of the ICP algorithm.

Rarely can significant performance gains be made without compromises. Applying the above modifications has given rise to a number of limitations, which are fully explained later in 5.4.3.2.

The core functionality of the developed algorithm is shown in Figure 5.15, and fully listed in Appendix B on page 189. The core algorithm functions in much the same

way as the initial ICP algorithm shown in 5.4.1.1 on page 91, with a few significant modifications.

The developed algorithm can be described in a similar way to that of the original ICP algorithm in 5.4.1.1 on page 91 by the following steps:-

1. Firstly, the two point-clouds are compared and the nearest neighbour of each of the scan/model points (m_i) is found in the map dataset (d_i) and the respective mean distances c_m,c_d are calculated using:

$$c_m = \frac{1}{N}\sum_{i=1}^{N} m_i \qquad c_d = \frac{1}{N}\sum_{i=1}^{N} d_i \tag{5.16}$$

2. Given that the rotation of the two point-clouds is corrected through the orientation sensor only the translation needs to be corrected. This can be found through using: $\mathbf{t} = c_m - c_d$

3. Step 1 and 2 is performed for a set number of iterations, unless the translation $\mathbf{t}$ is found to be erroneously large at which point the scan is discarded and the algorithm waits for new data.

4. The error function is then calculated. Given the simplified error function calculated earlier in Eqn. 5.14 and 5.15 on page 92:

$$E(\mathbf{R},\mathbf{t}) = \frac{1}{N}\sum_{i=1}^{N} \|m_i - (Rd_i + t)\|^2 \qquad N = \sum_{i=1}^{N_m}\sum_{j=1}^{N_d} sgn\,\omega_{i,j} \tag{5.17}$$

   This can be simplified further given that the rotation matrix $\mathbf{R}$ is no longer used:

$$E(\mathbf{t}) = \frac{1}{N}\sum_{i=1}^{N} \|m_i - (d_i + t)\|^2 \tag{5.18}$$

   Instead of calculating the error function for each iteration the error is only computed before and after the iterative stage. If the error suggests an incorrectly aligned scan the algorithm skips the scan and waits for new data.

Through using uniform weighting (ω) processing can be reduced further as N will remain constant between scans.

5. If a significant movement is detected since the last update of the map dataset (d_i), in this case ±0.2m height and 0.5m laterally, the aligned scan data (m_i) is added to the map dataset (d_i).

5.4.3.2 Limitations

The performance gains of the developed algorithm do, however, give rise to compromises:-

By using an *approximate* nearest neighbour search, as opposed to a *true* nearest neighbour search, errors are introduced into the scan-matching process. Parameters can be set in the FLANN algorithm which can be used to specify the levels of approximations used. More detailed searches can be made but at the cost of increased computational time. Tests discussed in 6.4.3 on page 128 showed that the error is acceptable for use in-flight, however for off-line processing, where detail and accuracy is required, a true nearest neighbour search should be used.

Reducing the number of points used and stored during SLAM process significant memory and performance gains are achieved. However, it will result in loss of detail, adversely affecting the accuracy and robustness of the scan-matching process. Through experimentation and testing shown in 6.4.2 on page 126 it was found that not all of the 1080 points from the LiDAR scanner needed to be used. Instead it found that similar and sufficient accuracy and robustness could be achieved through using every *fifth* (216) points (for the likely environments that this UAV is to be exposed to).

Removing the function which corrects for rotation considerably lowered the iterative processing time, however its complete removal is questionable. The developed

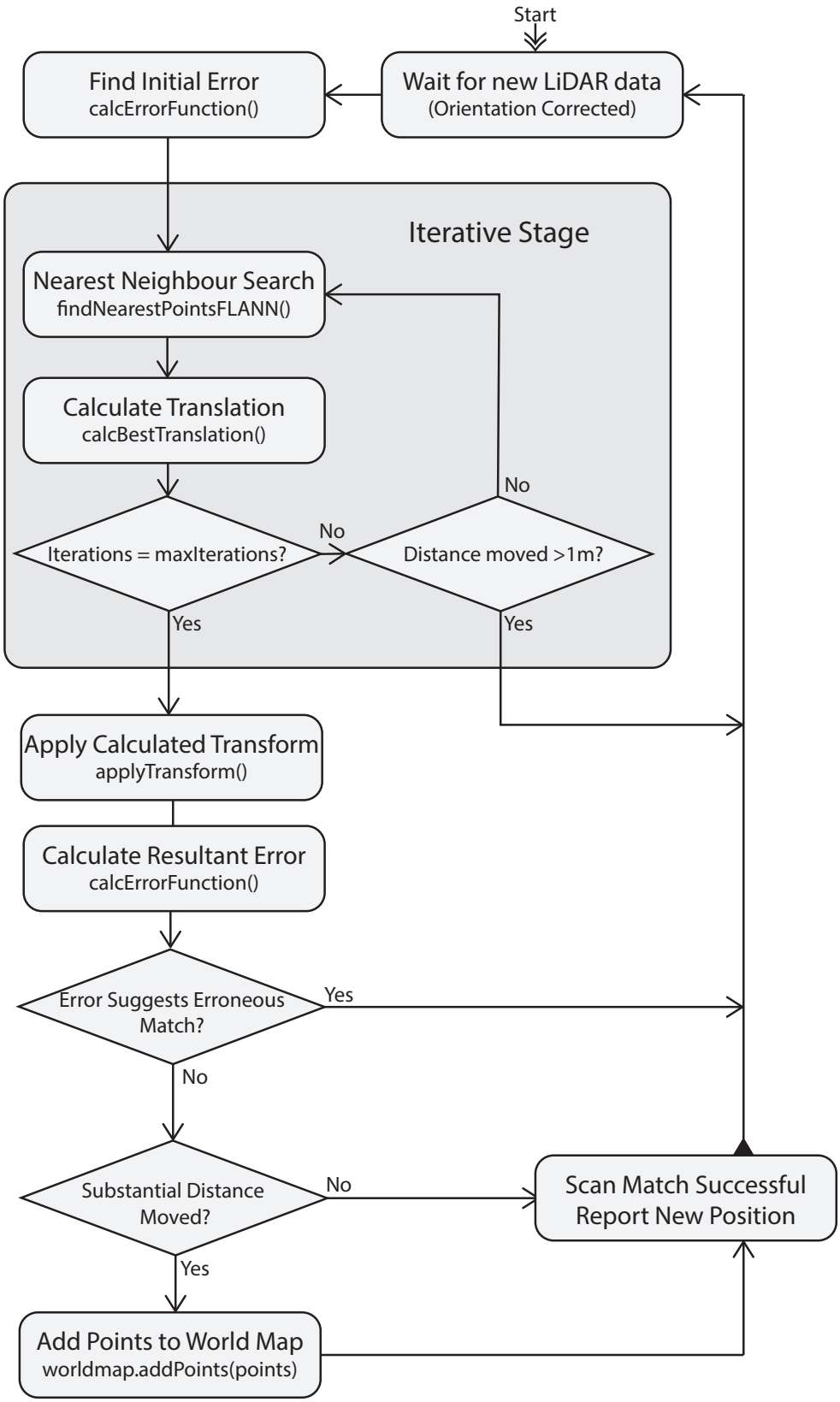


Figure 5.15: Flow diagram depicting the developed SLAM algorithm. For full listings see Appendix B on page 189

SLAM algorithm only solves translational drift and does not correct for any rotational component, relying solely on accurate, drift free heading data from the IMU. Accurate heading from the IMU, however, is not guaranteed and can be prone to significant deviation (drifting) depending on the sensor used and its placement on the UAV. If significant drift occurs, the integrity of the map will become compromised, causing localisations based on the previous map to be erroneous. As discussed and demonstrated in both the sub-system testing (Section 6.1.2 on page 118) and final flight tests (Chapter 7 on page 144), this did not pose a problem during testing. However, if further development is planned, it may prove beneficial to implement a periodic routine for checking the rotational alignment instead of blindly agreeing with the IMU sensor data.

By moving the error function out of the iterative section of the algorithm and relying on a before and after scan-matching comparison, it no longer becomes possible to check whether each iteration is converging or diverging from a solution or determining whether further iterations are necessary. Instead a fixed number of iterations are used, derived through testing to give the best compromise between convergence and performance.

5.4.3.3 Post-Processed vs. Real-Time Mapping

As one of the requirements state that the data collected by the UAV should be able to be produced into high-density point-clouds, which can later be used for inspection, two versions of the proposed mapping algorithm exist. The *real-time* maps are generated using the methods proposed above with the focus on high-speed robust localisation. The point-cloud generated by this algorithm, although sufficient for establishing navigational information, is not sufficient for inspection purposes. Instead a modified version of the algorithm was developed, which analyses the logged data through *post-processing* the data, both to more accurately verify the assumptions made by the proposed algorithm but also to provide these highly detailed point clouds.

The offline mapping algorithm is identical at its core to the real-time algorithm, however, as processing time is no longer restricted, the algorithm utilises every point from each scan, uses octree optimisation instead of FLANN for increased accuracy and functions on logged scan data (stored locally on-board the UAV during the flight). The UAV's logged data can also be exported to commercial grade point-cloud registration software if needed and available. No such software was used in the processing and development for the point-clouds shown in this thesis.

5.5 Ground Station

Shown in figure 5.16 is the complete ground station. The ground station comprises of two distinct components - communication equipment for relaying data to and from the operator station, which allows the operator to effectively control the UAV.

The communication equipment comprises of a two way data-link allowing digital data to be transferred between the UAV and the operator station, discussed in 5.5.3 on page 111, and a separate real-time video feed discussed in 5.5.4 on page 112. Although shown to be in close proximity in the photo, in reality the radio equipment can be placed at a distance from the operator's station through the use of a tether. This allows the radio equipment to be placed inside the building being flown in to allow for a clearer signal, while allowing the operator to keep their distance if required.

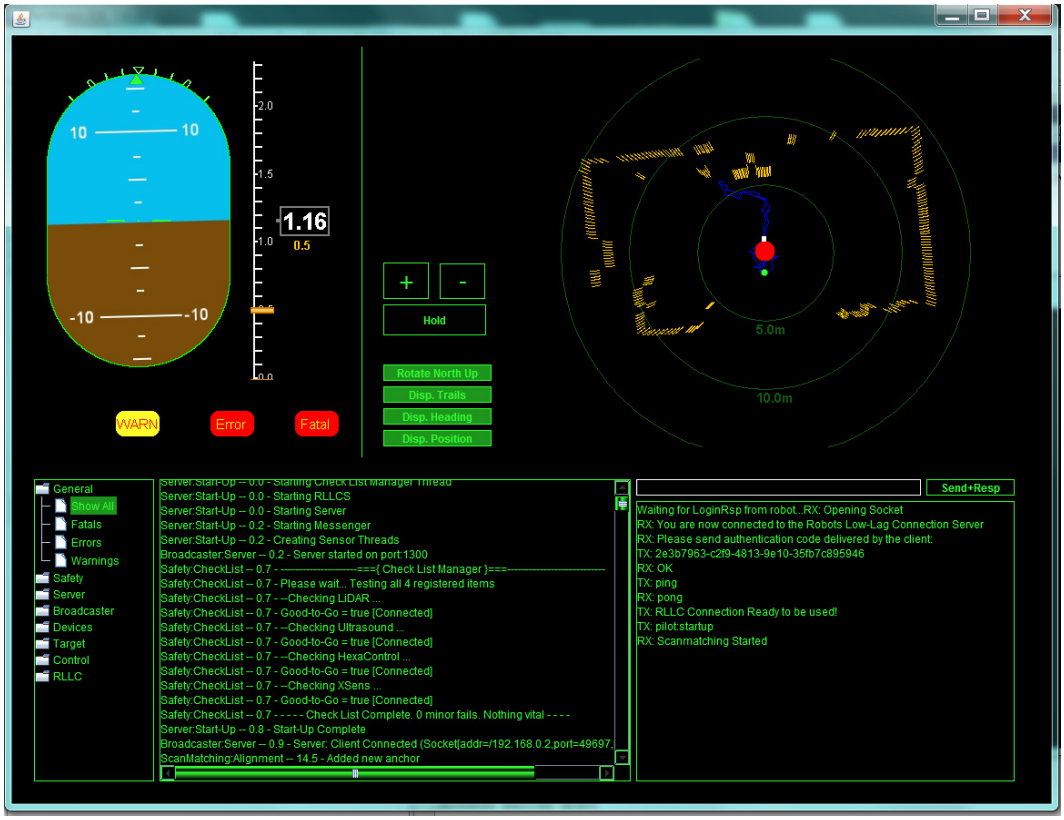
The operator's station comprises of the equipment available for the operator to help maintain situational awareness and to send commands to the UAV. A laptop is used as the primary source of information and control as discussed in 5.5.1 on the following page and utilising a popular game controller for easy, intuitive control (see 5.5.2 on page 110). The video feed is currently shown on a pair of video goggles shown in the centre of figure 5.16. It can be displayed on a standard television if more than one operator is required to see the footage.

Figure 5.16: The ground-station. Showing the laptop, controller, WiFi router, video receiver and goggles.

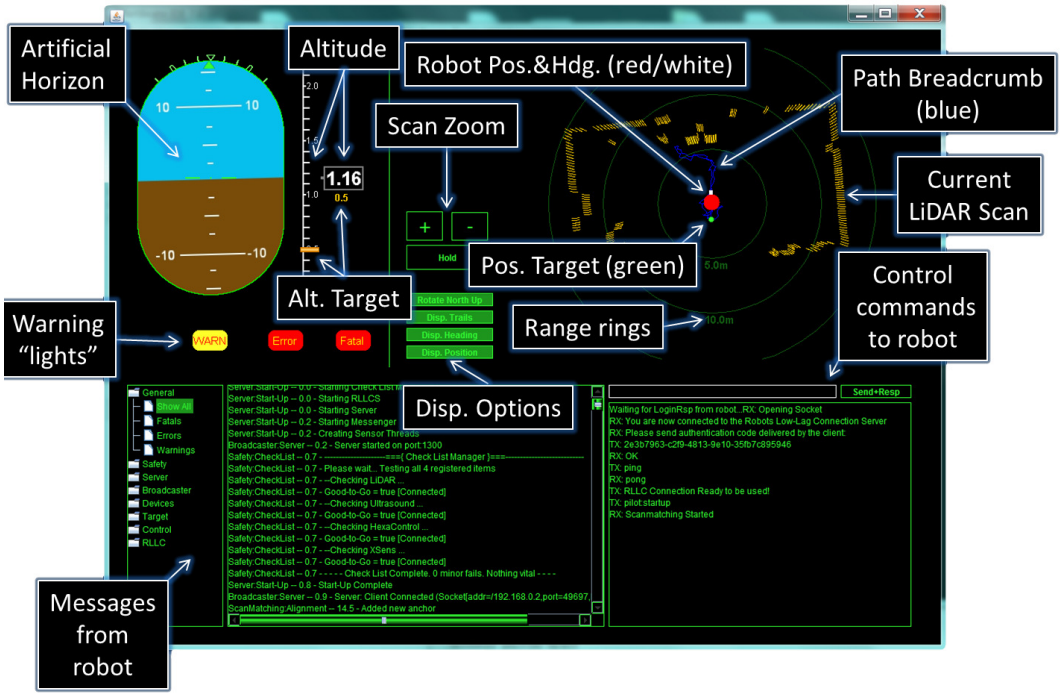
5.5.1 Graphical User Interface

The operator's laptop runs custom software, also developed for this project, which is used to interface with a server running on-board the UAV and display the necessary information in an easy to interpret form to the operator. The designed *Graphical User Interface* (GUI) is shown in figure 5.17. The GUI has been designed to allow fast dissemination of the UAV's system status', along with the rapid access to commonly requested functions.

The GUI can be split into two distinct sections. The upper half showing primarily the current position and orientation and the lower half showing more in depth messaging.



(a) Example screenshot of the GUI while in-flight



(b) Explanation of the various components of the GUI

Figure 5.17: The Graphical User Interface

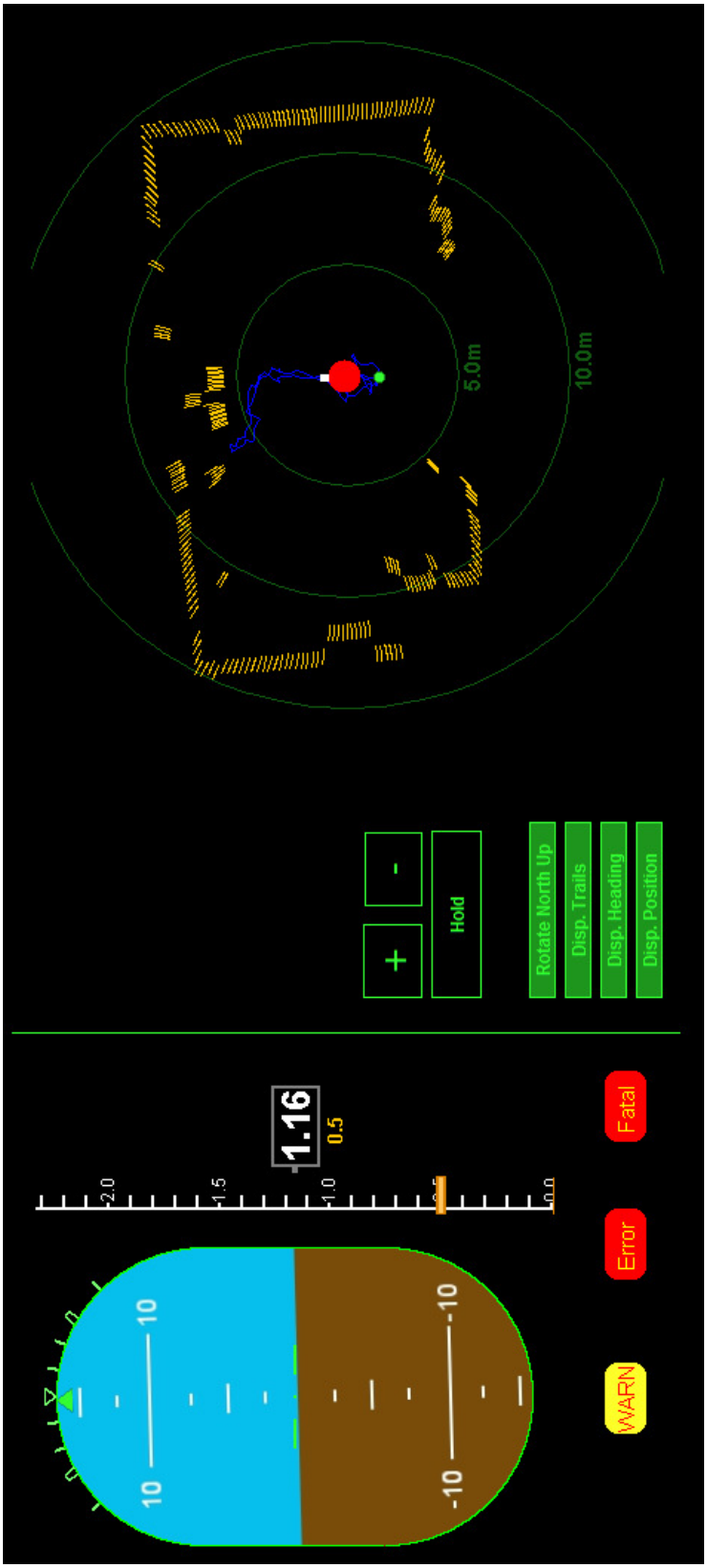


Figure 5.18: Enlarged version of figure 5.17a detailing the upper section of the GUI

5.5.1.1 Upper Half of the GUI

Figure 5.18 shows a detailed view of the upper half of the GUI. The upper half is designed to allow the operator to at-a-glance get full situational awareness of the UAV's present state, position and it's target goals. This is achieved through the use of a few "virtual" instruments (from left to right):-

1. An artificial horizon, updated with current data from the orientation sensor.
2. A slider both graphically and numerically displaying the current height of the UAV, along with the target height overlaid in orange.
3. Three warnings lights, each with a unique colour/flashing combination notifying the operator of detected failures and faults. If clicked it will highlight the error on the lower half of the GUI (see 5.5.1.2). The three levels are: Warning, a non critical process has encountered a problem. Error, a critical system has encountered a problem. Fatal, a critical system has encountered a potentially unrecoverable fault.
4. A top-down two-dimensional interactive zoomable map is used to display real-time LiDAR data, heading orientation, "breadcrumb track" of previous movement and the current target position. To allow for fast and easy navigation of the UAV if the map is clicked, the software calculates the physical position of the computer's cursor in the UAV's co-ordinate system and instructs the UAV of its new waypoint. This allows the operator to essentially laterally navigate the UAV through simply clicking on the "map".

5.5.1.2 Lower Half of the GUI

The lower half of the GUI, shown in figure 5.19, is more of a secondary interface, displaying current system status, debug information and a log of commands sent to the UAV, as well as the option to manually type non-standard commands if required.

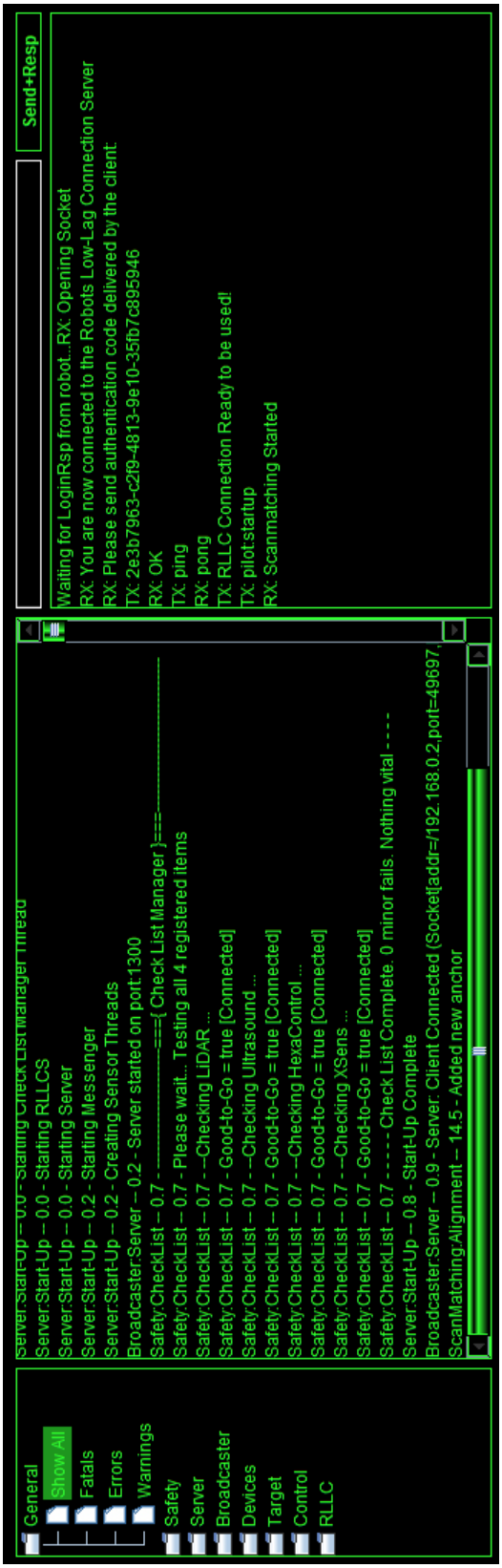


Figure 5.19: Enlarged version of figure 5.17a detailing the lower section of the GUI

The messenger system (left) displays all the messages generated by the various classes and systems on-board the UAV and allows for rapid filtering of messages either by "type of error" or by name. This allows the operator to stay informed in detail about the specific status of the UAV in case an error or anomaly is detected.

The *Robot Low Lag Connection* (RLLC) was initially developed by the WMR team[70]. The RLLC allows for the sending of commands from the robot through a plain-text TCP/IP connection. This not only allows the operator to inspect the commands being sent, but also manually type commands which may be non-standard or not yet implemented into the GUI.

5.5.2 Game Controller

By using a dedicated game controller with a fixed layout for various commonly used functions, it becomes possible for the operator to intuitively control the UAV through haptics alone. This allows the operator to become visually immersed in other tasks, such as monitoring the live video feed while still being in-control of the UAV.

By moving the common functionality from the GUI to the game controller, the GUI's clutter and visual complexity can be reduced, further aiding the operator. The operator's interaction with the GUI has been reduced as far as possible due to the concentration and time needed to issue instructions. For example, if a game controller were not used and the operator was focusing on monitoring the video feed, a simple instruction such as rotate would require the operator to look over to the laptop, find the cursor, move the mouse to the required button, click and then return his attention to the video feed, thus requiring a lapse of situational awareness for each instruction. The use of a controller always-in-hand allows for a fast and haptic method of sending these common commands without the need to break eye contact.

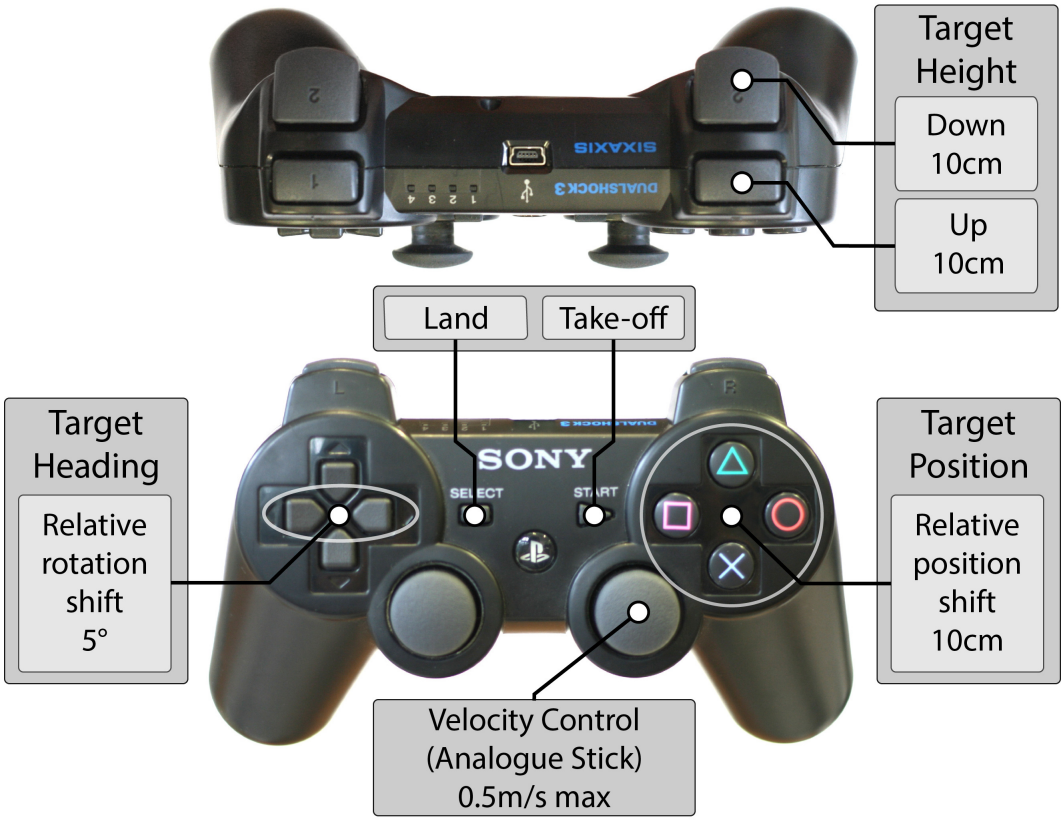


Figure 5.20: Layout of the Game Controller

The layout of the game controller is displayed in figure 5.20, which allows for basic control of the UAV. For "long distance" navigation where the UAV is clear of obstacles, the use of the GUI's point-and-click navigation, discussed in 5.5.1.1, is encouraged. Although taking slightly longer to issue the instruction, a more precise waypoint can be chosen with reference to the LiDAR data returned from the UAV.

5.5.3 Two-Way Data Communication

For ease of use and cost effectiveness all digital data and control commands are transmitted through Wi-Fi, although a multitude of different digital transmission

systems could be used. A Wi-Fi router is located at the ground station which the on-board computer connects to. The operator's computer can be connected either using a wired or wireless connection to this router enabling communication between the operator's computer and the UAV. To reduce latency and bandwidth only minimal data, needed to populate the GUI and instruct the UAV of its next waypoint, is sent. The remaining data is logged and stored locally.

Before deployment for long (>50m) distance missions, much like the live video feed, the Wi-Fi will need to be upgraded to a different transmission method or towards a high-power licensed Wi-Fi router to get the range required.

5.5.4 Live Video Feed

As the on-board video feed is not being used to enhance the robots abilities, but solely to increase the situational awareness of the operator, the video system has been designed to be completely separate from the other systems on the UAV. The benefits from this are two fold, firstly, it helps to conserve bandwidth and to minimise latency on the Wi-Fi communications. Secondly, if the UAV were to be commercialised, licensed high-power transmitters would be required in order for it to operate at a useful range from the ground-station, whereby costs would be dramatically reduced by using a dedicated simplex video transmitter combined with a lower bandwidth duplex digital transmitter.

To minimise crosstalk and interference with the other transmitters and receivers on-board the UAV the video system should ideally operate in a frequency band not in use by other on-board systems. There are many hobbyist video transmitters designed for RC models, which are as compact and lightweight as possible. These are available in range of "public" frequency bands (900MHz, 1.2GHz, 2.4GHz and 5.8GHz). However, most are developed for the US market and many are illegal for use in Europe and the UK due to differing regulations with regards to public broadcast bands. One example is the popular 900MHz and 1.2GHz video systems,

which in Europe is used by GSM mobile phone network. In the UK the only publicly open frequencies, which these products have been developed for, are in the 2.4GHz and 5.8GHz band. As the Wi-Fi router and the safety controller functions in the 2.4GHz band, the 5.8GHz band was chosen, even through the higher frequency is more restrictive towards line-of-sight operation and suffers greater from radio shadows and multi-path propagation inside buildings [81].

A commercially developed hobbyist 5.8GHz video transmitter and receiver designed specifically for RC aircraft were purchased. The transmitter power is limited to 25mW due to OFCOM licensing regulations[82]. In light of the potential issues with localised radio shadows and multi-path propagation a more advanced "diversity" receiver was ordered. The "YellowJacket 5.8 Pro Diversity Receiver" utilises two aerials and receivers and internally monitors the signal strength of each receiver, which allows it to automatically toggle to the receiver with the strongest (and usually) the clearest signal. The weakness is, that it only monitors signal strength, but doesn't check the signal itself, meaning that if there is a strong interfering transmission on a similar frequency, the receiver may mistakenly lock on to the interfering signal [83].

5.5.5 Safety Controller

To aid testing and development of the autonomous features of the UAV and also to increase safety, the UAV platform not only accepts input commands from the on-board computer but also from a model RC controller (see figure 5.21). This enables the UAV to be flown manually when needed or even relinquishing certain controls in order to test individual systems.

On top of the standard throttle, yaw, pitch, roll channels three extra channels were used:-

- Emergency-Stop: If this switch was engaged the motors on the UAV would be switched off regardless of the position of any of the other inputs.

Figure 5.21: The model RC controller used (Spektrum DX7)

- Level of Control: This is a three position switch and varies the amount of control given to the PC.

  Throttle - Only throttle is controlled by the computer, yaw, pitch and roll controlled manually.

  Position - Yaw, pitch and roll controlled by the computer, throttle controlled manually.

  Full Auto - No manual control, computer controls all channels.

- Take Control: Toggles between the control modes above and full manual control.

Chapter 6

Sub-System Testing

When testing a newly developed system it is important not only to test the system as a whole, as discussed later in Chapter 7, but also to test the individual sub-systems from which it is built. Through testing and verification of the function of the individual sub-systems greater reliability and performance can be obtained. This chapter aims to verify any assumptions made during the development and to prove the proposed SLAM algorithm.

6.1 Sensor Performance

Throughout the *Sub-System Design* Chapter (Chapter 5) a number of assumptions were made regarding the functionality, performance and reliability of certain sensors. There were two primary assumptions. Firstly, reliable height information can be obtained through the use of a downwards facing SoNAR sensor and that, when the SoNAR sensor is out of range, a pressure based altimeter could be used in its place (assumed in 4.4 on page 65). Secondly, that the XSens orientation sensor provides accurate and low drift data regarding the UAV's heading during flight (assumed in 5.4.3.1 on page 99).

6.1.1 Height Detection

Accurate and reliable height information is critical to enable the correct function of the UAV's control system and the SLAM algorithm. The proposed UAV solution

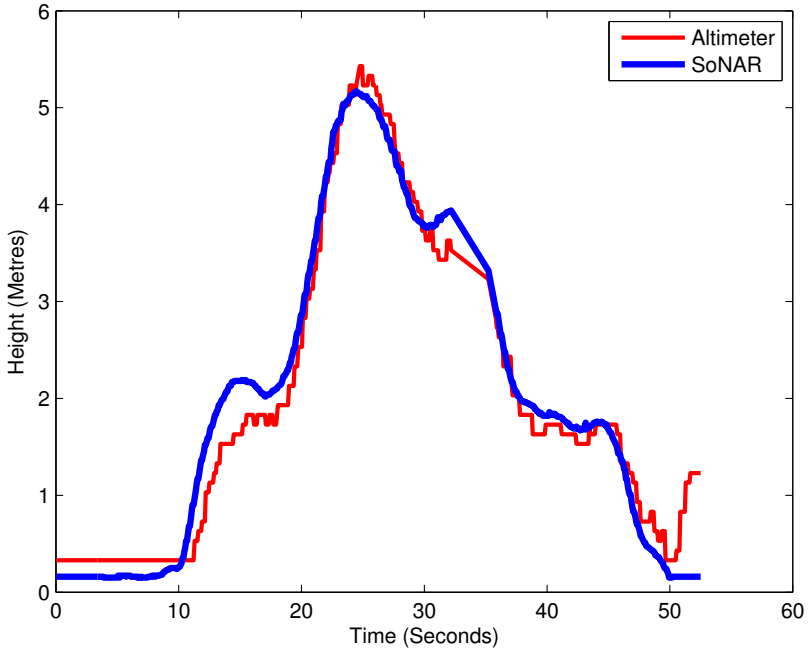


Figure 6.1: Comparison between SoNAR and altimeter height information, climbing and descending over level ground

utilises two methods for detecting the height of the UAV as discussed earlier. When the UAV is less than 5-6 metres above the ground the downwards facing SoNAR sensor is used, when above this range the altimeter is used in its place.

Data was collected from both the SoNAR and pressure altimeter during flight to ensure that the data provided was coherent and as expected. Initially there was some doubt of the quality of the data produced. For instance, the pressure altimeter becomes unreliable when exposed to turbulence which may occur from the rotor blades and the SoNAR's ultrasonic pulse may be interfered with by the noise from the rotors and the associated airflow. The altimeter's sensitivity was also disputed due to the low differential in atmospheric pressure when climbing/descending short distances.

Figure 6.1 shows a short flight in which the UAV is flown over level ground to test the performance of the two sensors. The SoNAR data closely resembles the observed

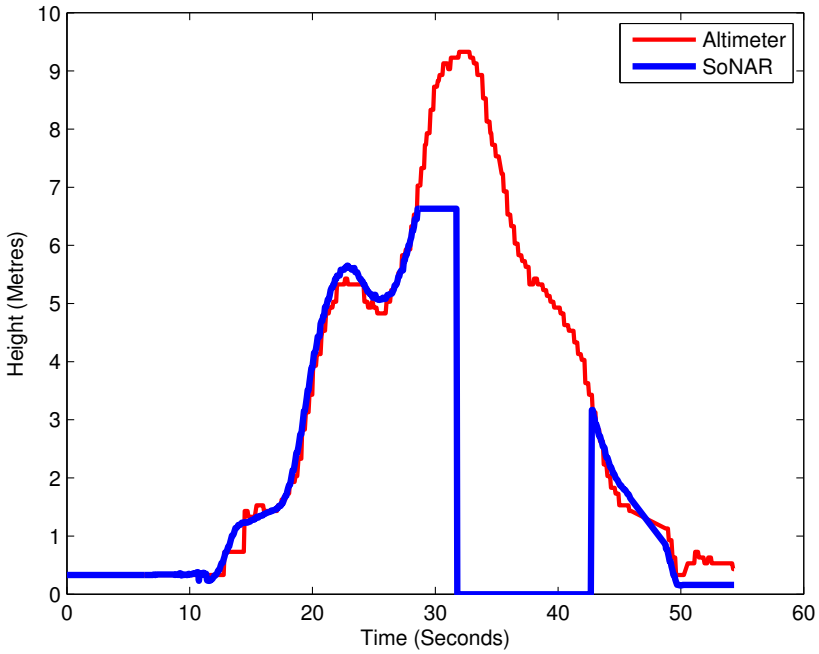


Figure 6.2: SoNAR response in an "out-of-range" condition

height profile of the UAV during the test and remains functional throughout. The altimeter closely follows the SoNAR plot, however with some slight discrepancy and quantisation of the data. The reported height of the altimeter during landing however is erroneous, most likely due to ground-effect as the UAV is in close proximity to the ground (<20cm), which creates a "cushion" of higher pressure air causing incorrect readings.

For this reason the proposed UAV utilises the SoNAR sensor when flying close to the ground, less than 5-6metres, to allow for accurate height detection of the ground below for take-off and landing, as over time there may be slight pressure differences in the room causing the altimeter data to drift. This drift does not significantly impair the UAV when at height, however can pose a significant issue when attempting to land.

To demonstrate that the SoNAR will re-aquire a "lock" on the height of the UAV another test was performed, whereby the UAV was flown outside the range of the

SoNAR, then descended to a landing. A sample of the results of these tests is shown in Figure 6.2, and shows that that the UAV needs to be flown well under the SoNAR's maximum range in-order to re-aquire the signal.

A problem arises, however, when the UAV is not flown on level ground and the SoNAR sensor is being used, as the SoNAR sensor will report the height above object below the UAV, which may not necessarily be the floor. To counter this the UAV checks for any large fluctuations in the SoNAR's reported height. If a sudden step of more than 30cm is detected, then an offset is set which adjusts for the height of the object allowing the UAV to some extent not to be affected when overflying cluttered areas, as demonstrated in figure 6.3.

One scenario which the UAV does not presently account for, which was not stated in the requirements, is a ramped change of height of the floor. A ramped change in height would not be detected by the step filter discussed above, and the only way to detect this scenario would be to contrast the altimeter and SoNAR measurements during the flight.

6.1.2 Heading Data from the Orientation Sensor

To test the assumption that the UAVs approximate heading can be derived solely through the use of the XSens orientation sensor and that the UAV's vibration and electromagnetic interference does not compromise the sensor, a short test flight was conducted whereby the UAV was briefly flown with large heading fluctuations, shown in figure 6.4. By landing the UAV in the same orientation that it took off from, the magnitude of the heading drift can be seen, which is in this case is negligible.

6.2 Communications

Initial outdoor tests of the video system showed acceptable range and clarity. However, later tests inside the test halls, which are large open halls, usually with metal

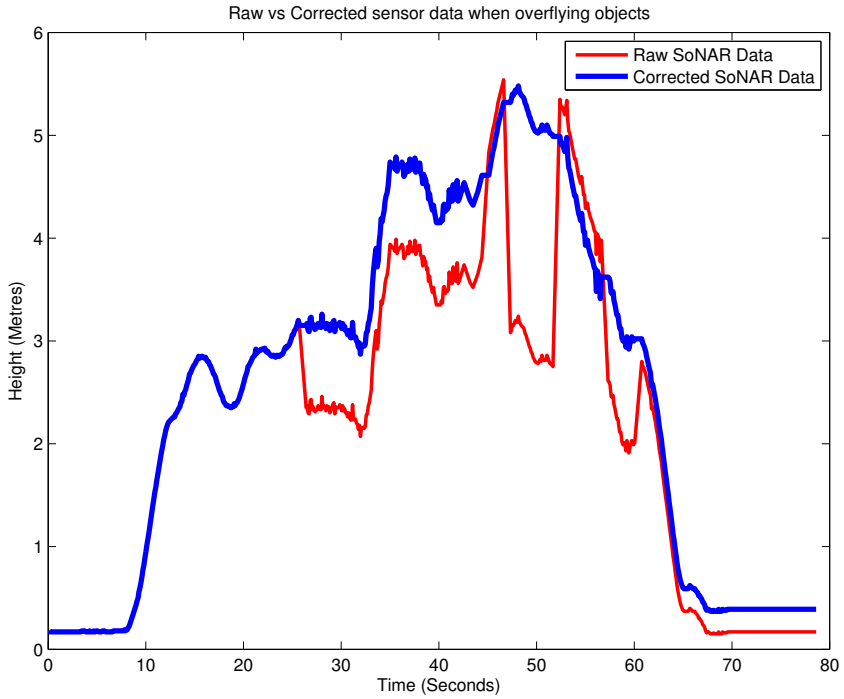


Figure 6.3: Utilising step detection to lessen the impact of overflying an object when using the SoNAR sensor

roofing and heavily reinforced concrete walls, significantly reduced the operating range to a point where it was unusable in most cases. Occasionally only getting an effective range of 5 – 10metres before the video was unrecognisable due to distortion and noise (analogue video transmission).

It was believed the cause of the drastically reduced range still was the result of heavy multi-path interference, where the signal was reflecting off the walls and ceiling before returning to the receiver.

A number of solutions to reduce the encountered interference were researched, but the common solutions required the current transmitter and receiver to be replaced with more expensive, high-end devices employing signal error detection and recovery or more exotic modulation methods. One simple solution that could easily be implemented was to change the polarity of the antennas of the system. The standard “rubber duck” antennas (see figure 6.5) that are provided as part of the system

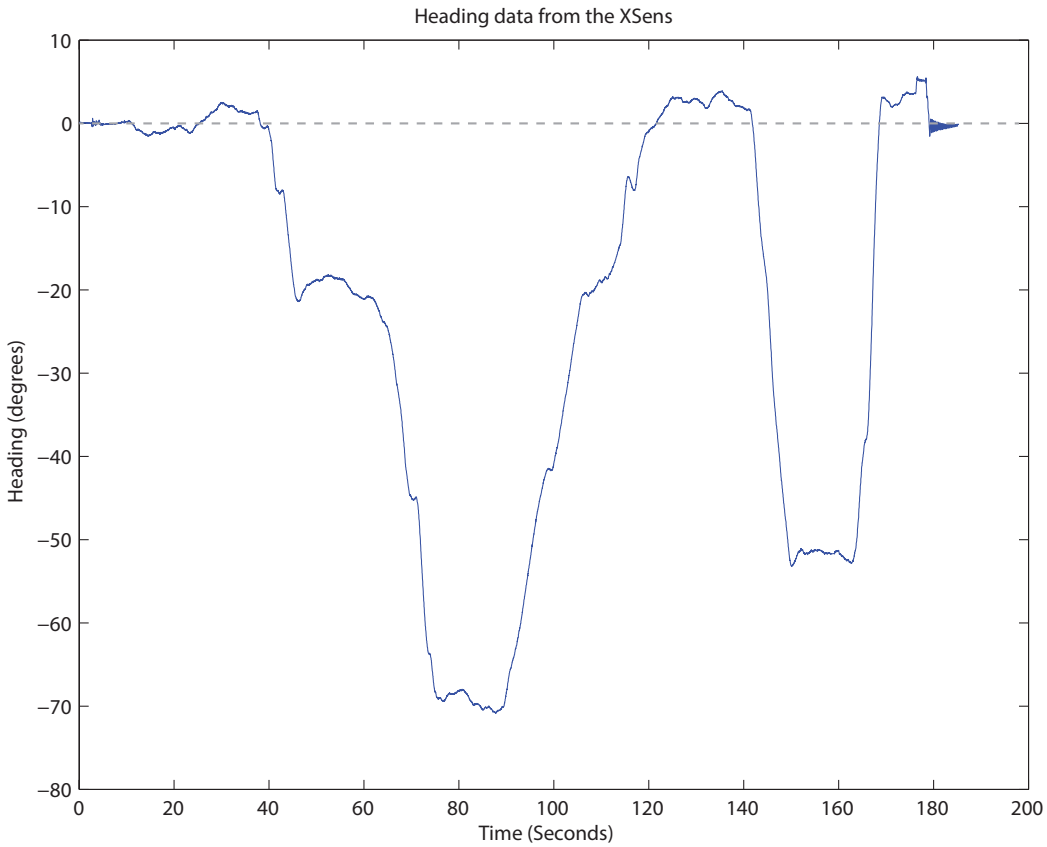


Figure 6.4: A graph showing the heading data outputted by the XSens during a short flight where the take-off and landing was in the same orientation.

Figure 6.5: Linear "rubber duck" antenna (left), cloverleaf circular polarised antenna (right)

transmit a mostly linearly polarised signal This has a further disadvantage not yet discussed. where, if the model aircraft is banking heavily, the polarisation angle between the receiver and transmitter will vary greatly causing signal loss. Fortunately, as the UAV is generally in level flight this effect is minimal. This is also the case with light if a radio signal is reflected off a surface. Depending on the angle of incidence the reflected signal will become either linearly polarised with respect to the angle of surface or loose its polarisation [84]. The solution is to use circularly polarised antennas as a circularly polarised signal rotates through time as opposed to oscillating in a particular direction (linear). Circular polarisation can either have a clockwise or anti-clockwise rotation, analogous to horizontal and vertical linear polarisation.

The benefit of using circular polarisation is that if the signals are reflected the polarisation will be changed and therefore be attenuated by the circularly polarised antenna at the receiver. This has not solved the interference problem, but has greatly increased the indoor range, typically to more than 30 metres. Figure 6.6 shows the improvement from using circular polarised antennas instead of linear antennas inside a workshop at approximately 15 metres range.

(a) Linearly polarised antenna

(b) Circularly polarised antenna

Figure 6.6: Comparison of video clarity from (a) linearly and (b) circularly polarised antennas inside a workshop at 15m range.

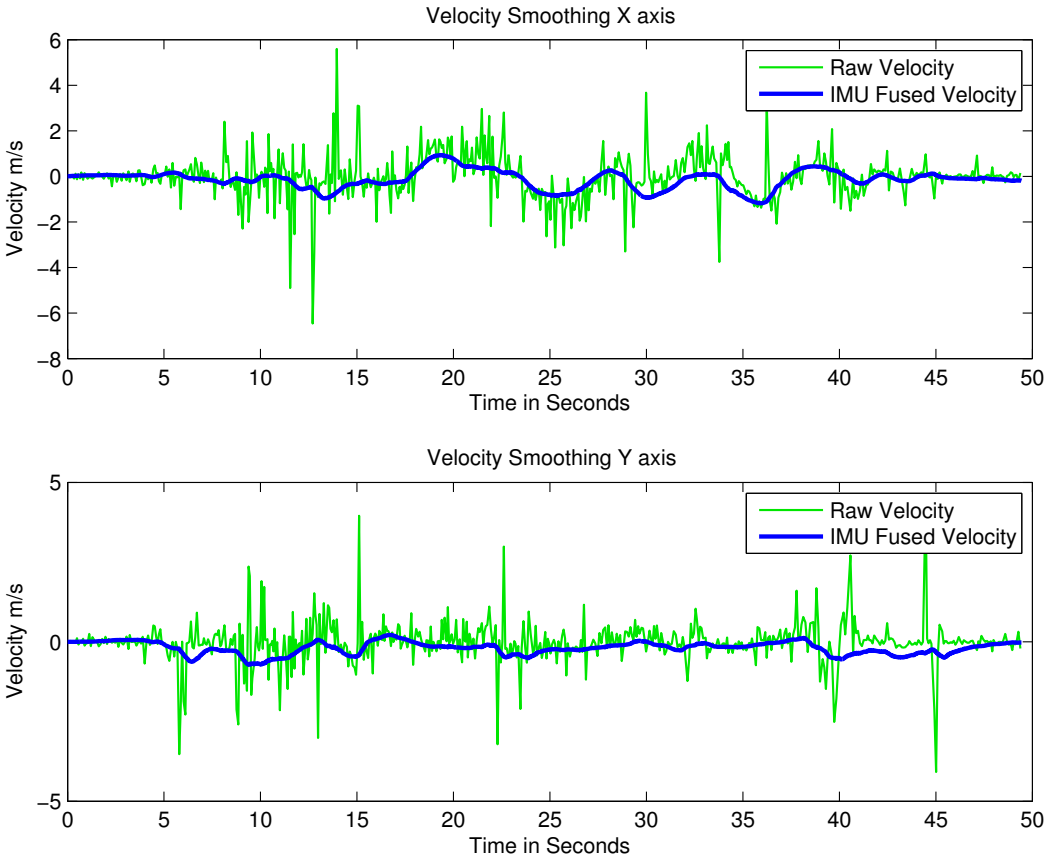


Figure 6.7: Comparison between raw data and fused velocity data produced by the Velocity Estimator

6.3 Velocity Estimator

The *Velocity Estimator* described in 5.3.4 on page 86 utilises data from both the IMU and the SLAM algorithm to produce an estimate of the current velocity of the UAV with significantly reduced noise. Figure 6.7 displays a sample of data collected from a test flight, it shows both the raw un-filtered data and the output of the IMU fused velocity. Simply using a low pass filter would introduce a phase-change in the resulting signal, which manifests itself as a time-delay, shown in figure 6.8. Initial tests showed that this time-delay introduced significant oscillations in the control algorithms. It was for this reason the *Velocity Estimator* was implemented.

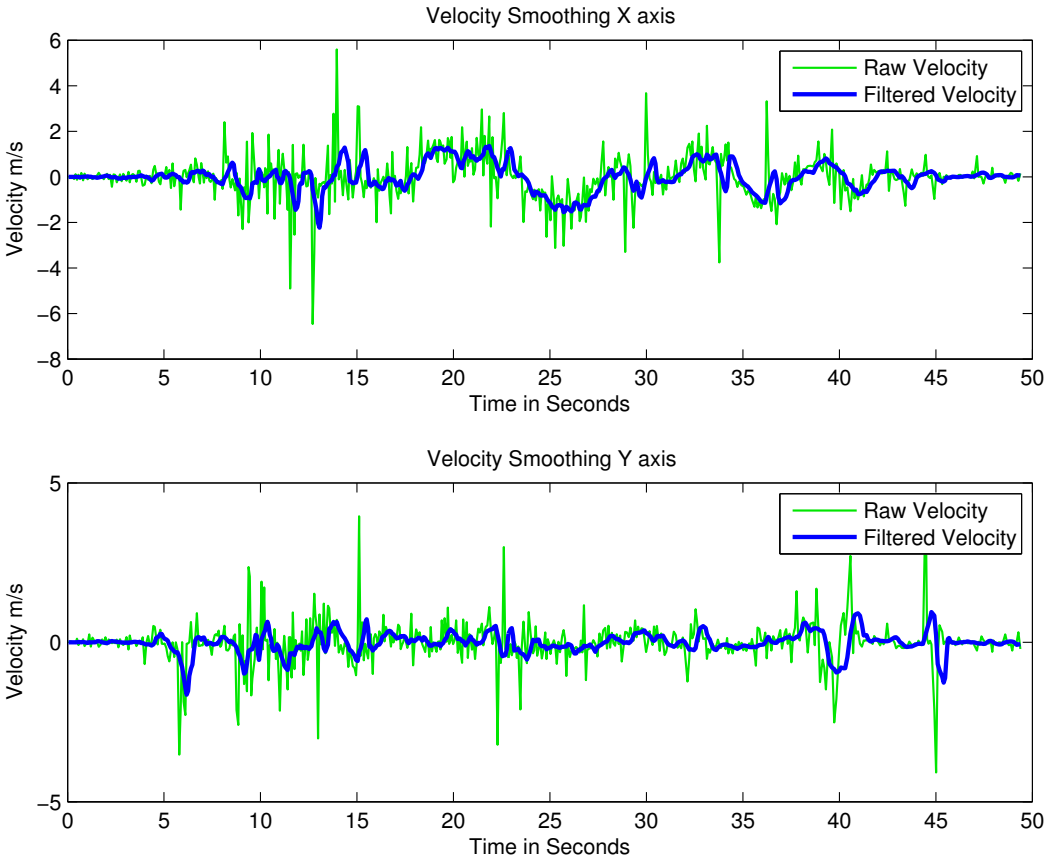


Figure 6.8: Comparison between raw data and using a simple weighted average filter to reduce noise.

Algorithm Function	Processing time
ScanMatching.ICP.**iterate** ()	6,396 ms (59.9%)
ScanMatching.ICP.**findSimilarPointstree** ()	4,484 ms (42%)
ScanMatching.ICP.**calcCostFunction** ()	1,160 ms (10.9%)
ScanMatching.ICP.**calcBestRotation** ()	708 ms (6.6%)
ScanMatching.ICP.**calcBestTranslation**()	34.0 ms (0.3%)

Table 6.1: Profiling of the iterative stage of the initial ICP localisation implementation, showing the total time to produce a small map.

6.4 Performance of the SLAM Algorithm

This section aims to demonstrate the performance gains achieved by the proposed SLAM algorithm described in section 3.3.2 on page 40. Verification of the reliability and robustness of its function in a range of varying scenarios is shown later in 6.5 on page 129 and during flight testing in chapter 7 on page 144.

6.4.1 Changes Made to the Original Algorithm

As mentioned in 5.4.3.1 on page 99 the original ICP algorithm's iterative stage comprises of a number of functions which has been removed in the proposed algorithm. A profiling of the original ICP algorithm's iterative stage during the process of building a small map can be viewed in table 6.1, the data of which was gathered through remotely profiling the algorithm onboard the UAV's using JAVA/Netbeans during a short flight.

As demonstrated in this figure the most process intensive task is the nearest-neighbour search (findSimilarPoints), followed by the error-function (calcCostFunction), rotation estimator (calcBestRotation) and translation estimator (calcBestTranslation). The error-function overheads are significantly reduced in the proposed algorithm as it has been moved out of the iterative stage of the algorithm and is now used as a before-and-after check to verify the success of the matching of each scan. The rotational estimator component has been completely removed and the efforts to reduce the nearest-neighbour overheads are discussed below in 6.4.2 and 6.4.3.

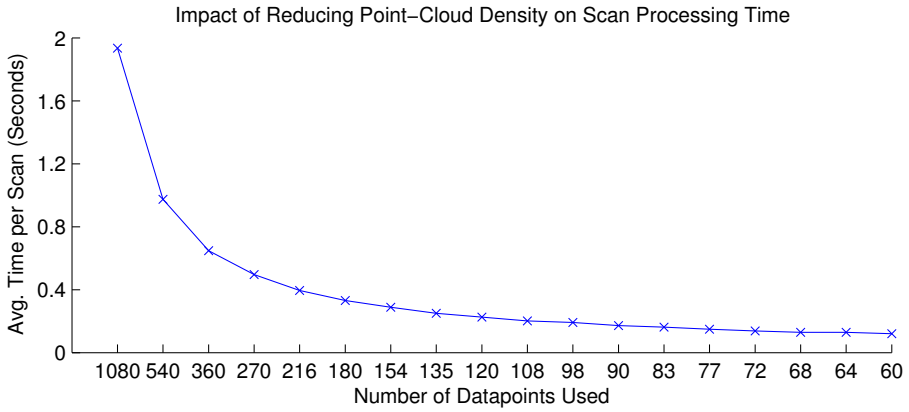


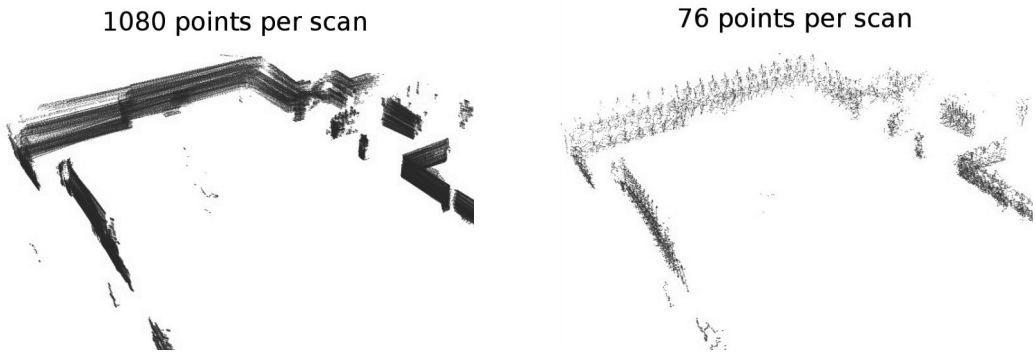
Figure 6.9: Number of points used vs. approximate processing time per scan (based on Octree).

6.4.2 Reduction of the Point-Cloud density

Reducing the amount of points used for both the scan matching and points stored in the map greatly decreases the processing time and memory requirements of the SLAM algorithm as depicted in figure 6.9. The proposed SLAM algorithm uses two methods to reduce the number of points that need processing.

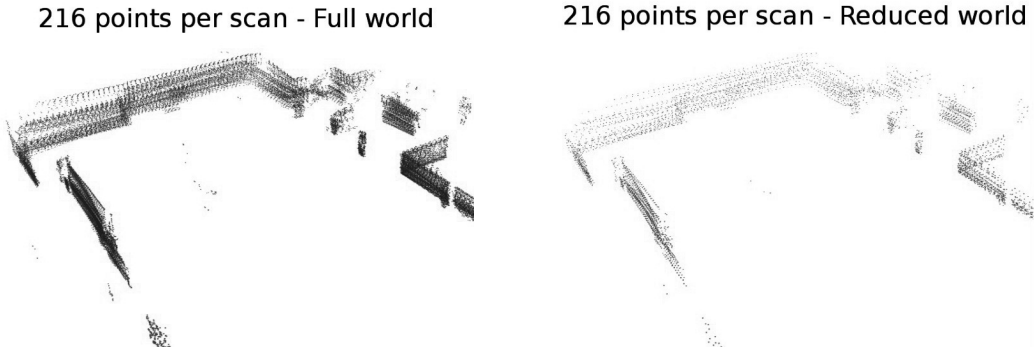
The first method is through globally reducing the number of points handled. This is done through limiting the number of points read by the LiDAR, a demonstration of this can be seen in figure 6.10. This reduction, however, is a balance between the required accuracy of the SLAM algorithm and the performance gain required. As points are removed from the scan, detail is lost and noise from object clutter becomes more difficult to distinguish.

The second method used in the proposed algorithm is reducing the number of points stored in the "world" map, thereby lightening the load for the nearest neighbour search algorithm. Not every new scan of the LiDAR contains data which would enrich the "world" map dataset, such as when the UAV is hovering in a set location. If data is added when significant movement is detected instead of after every scanmatch, the number of points automatically stored in the "world" map is reduced significantly, aiding the nearest neighbour searches to function optimally. An



(a) 1080 points per scan (b) 76 points per scan

Figure 6.10: Resulting point-cloud from using full (a) and reduced (b) dataset (width ∼30m)



(a) Full world (b) Reduced world

Figure 6.11: Resulting point-cloud from using all (a) or a subset (b) of the scans during map building

example of this is shown in figure 6.11, whereby the number of point used in each scan is kept the same. However, a new scan is only added when certain motion conditions are met. For this image and the flight tests discussed in the next chapter these conditions were set:-

1. Change in Height of ±0.2m
2. Change in lateral position of 0.5m.

6.4.3 Nearest Neighbour Search

As shown previously in figure 6.1 on page 125, the nearest neighbour search is the most process intensive part of the ICP algorithm and thus was a key focus during the development of the proposed algorithm. Initially an *Octree*[75] based sorting method was implemented due to its ease of use, however, during initial trials it was found to be too slow to be used in real-time on the on-board computer. It was for this reason that the *FLANN* algorithm[78] was investigated and implemented into the proposed solution.

FLANN performed approximately ten times faster than the *Octree* implementation used initially[68], which is demonstrated by data from a sample flight in figure 6.12. The data was obtained through flying the UAV for approximately one minute while logging all the sensor data. After the flight the data was then processed using the UAV's on-board computer. For each pass of the SLAM algorithm the number of points in the logged data was decimated, monitoring both the calculated end-position and the total time taken. The figure demonstrates a number of key aspects, firstly that the *FLANN* algorithm is significantly faster than *Octree,* however also shows the impact of implementing the point-cloud density reduction discussed in 6.4.2.

The drawback however with using an approximate nearest neighbour algorithm is that it finds the *approximate* nearest neighbour, not the true nearest neighbour, discussed earlier in 5.4.2.4 on page 96. This may not be significant when comparing high density point-clouds. However, as the density is reduced these approximate pairings can start to introduce errors, as a slightly mismatched nearest neighbour has a larger effect on the accuracy of the algorithm as a whole. Figure 6.13 demonstrates the effect of the point-cloud density reduction on the accuracy of the calculated final position after the SLAM algorithm had run.

As mentioned earlier in 5.4.3.2 on page 101, the proposed real-time algorithm uses every *fifth* point in the scan (i.e. 216 points total), which offers a significant re-

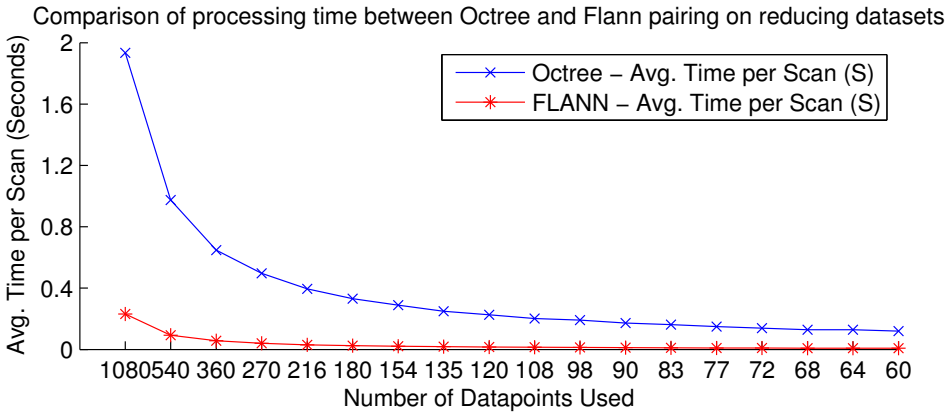


Figure 6.12: Comparison of processing time between Octree and FLANN pairing on reducing datasets [68]

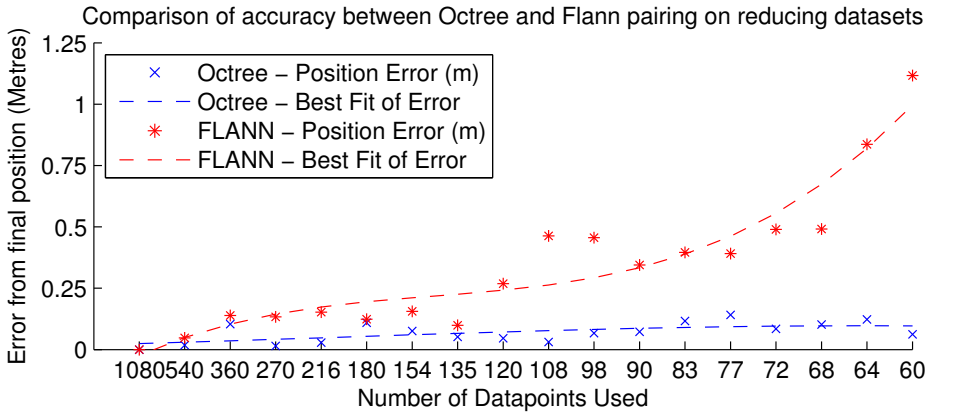


Figure 6.13: Comparison of error between Octree and FLANN pairing on reducing datasets [68]

duction in the processing time, however has enough points not to adversely affect the accuracy significantly in the process of mapping the environments the UAV was designed to be flown.

6.5 Testing of the SLAM Algorithm in Challenging Conditions

The SLAM algorithm has been tested by flying the UAV through as many visually dissimilar environments as possible. It was not possible to find a chimney of similar

dimensions to the one listed in the requirements, therefore no data is available for the SLAM algorithm's performance in a chimney environment. However, it is assumed that due to their simple regular geometry, usually remaining approximately the same through their height, that if the UAV functions well inside a cluttered three-dimensional environment, then it should function well within the simple and constant geometry of a chimney.

The following flights were performed by flying the UAV manually and not relying on its autonomous systems for control such as the flights performed in the following chapter "Flight-Tests". The reasoning behind hand flying the UAV is that it can be flown in smaller, more confined areas and also in environments where the SLAM algorithm is predicted to fail without risk of loss of control as is the case if flown autonomously. The goal of this section was to test the limits and performance of the proposed SLAM algorithm as opposed to the fundamental function of the UAV as a whole as demonstrated in the next chapter.

6.5.1 Cluttered Environment

Cluttered environments pose a challenging scenario for SLAM algorithms as the perceived environment can vary greatly between each successive scan. This can be caused by debris or small objects, which can be thought of as high frequency noise onto the scan data.

One of the most cluttered environments available was a workshop, a section of which shown in figure 6.14. The UAV was manually flown on a short flight above and around the cluttered area and the resulting real-time low-density map is shown in figure 6.15. The map shows that a coherent and visually similar map was produced and that the SLAM algorithm remained functional throughout despite the heavily cluttered environment.

Figure 6.14: Cluttered area of the workshop in which the UAV was flown.

6.5.2 Environmental Transition

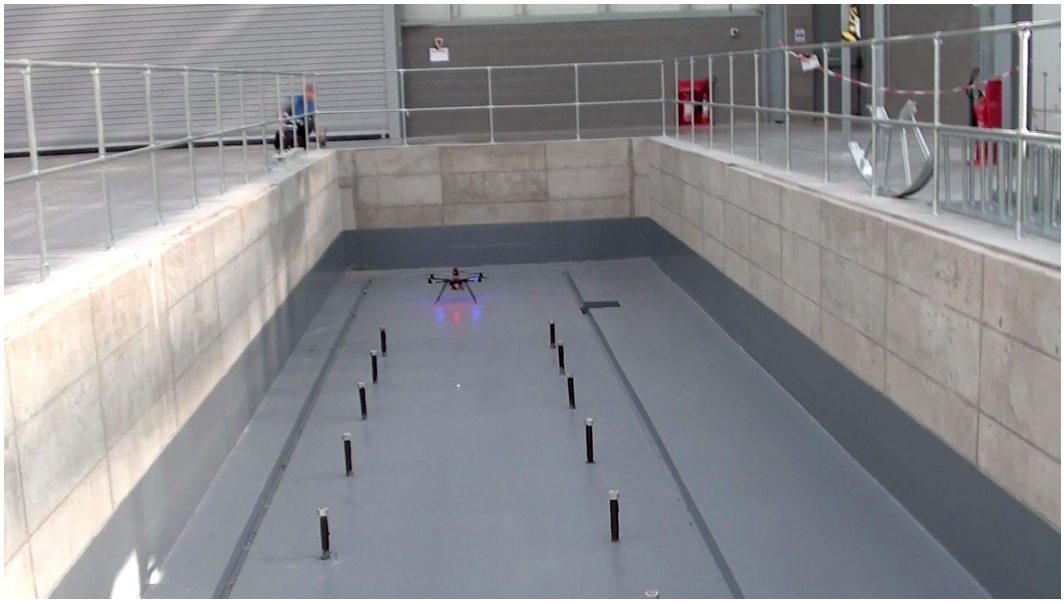
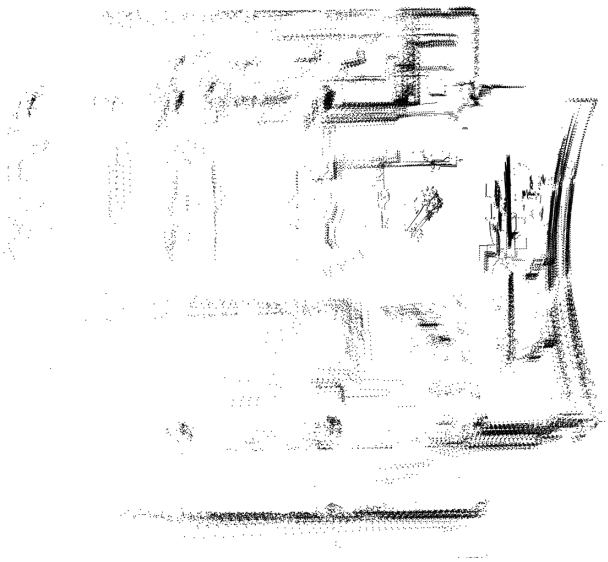
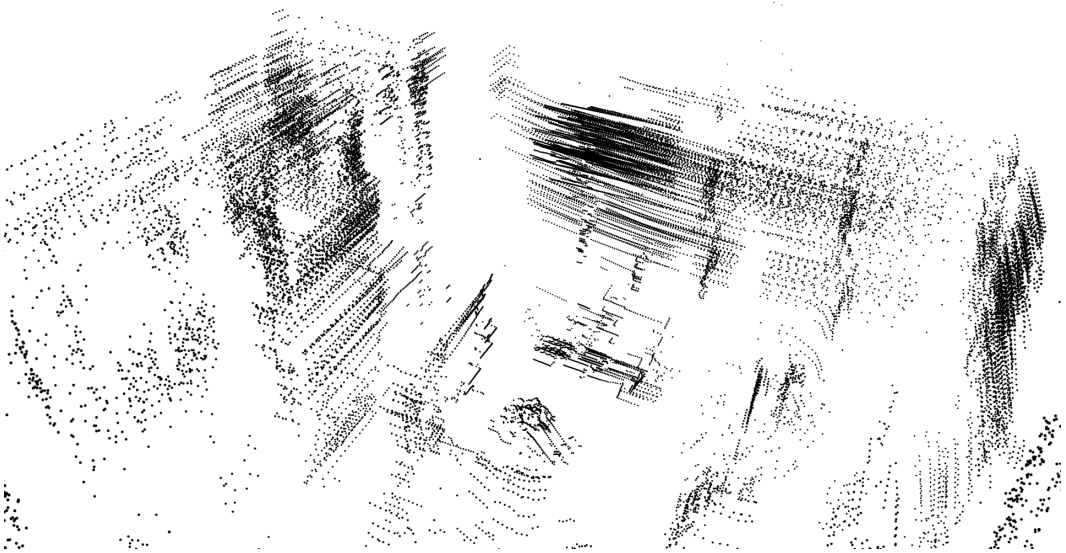


Figure 6.16: The pit used to test the SLAM algorithm's response to a sudden step change in its perceived environment



(a) Plan View



(b) Perspective View

Figure 6.15: High density, post processed maps of the workshop flight

To test the SLAM algorithm's response to a sudden step change in the perceived environment, where there would be a large apparent physical change in the structure's layout as it flew, the following test was devised. The test involved manually flying the UAV out of a pit, shown in figure 6.16, while the SLAM algorithm attempted to keep track of its location.

Figures 6.17 – 6.19 show the high-density post-processed and low-density real-time generated maps of the flight out of the pit, a video of the flight is available in Appendix D.2. There are three key observations to this test:-

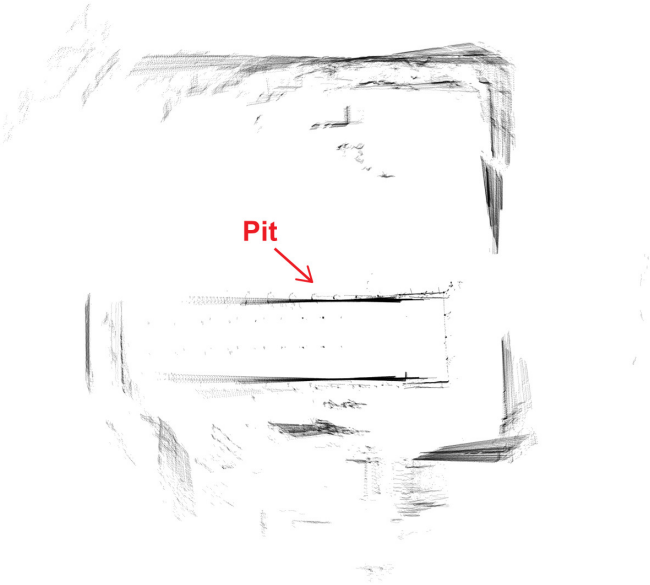
Firstly, figure 6.18 shows that the step detection applied to the SoNAR sensor (see 6.1.1 on page 115) functioned as designed, detecting the perceived height change when flying out of the pit and compensating to the change in height of the floor without adversely effecting the map.

Secondly, the mapping algorithm handled the sudden change flying out of the pit into the new environment without loss of significant or observable positioning, demonstrating the robustness of the proposed algorithm even in highly varying (yet static) environments.

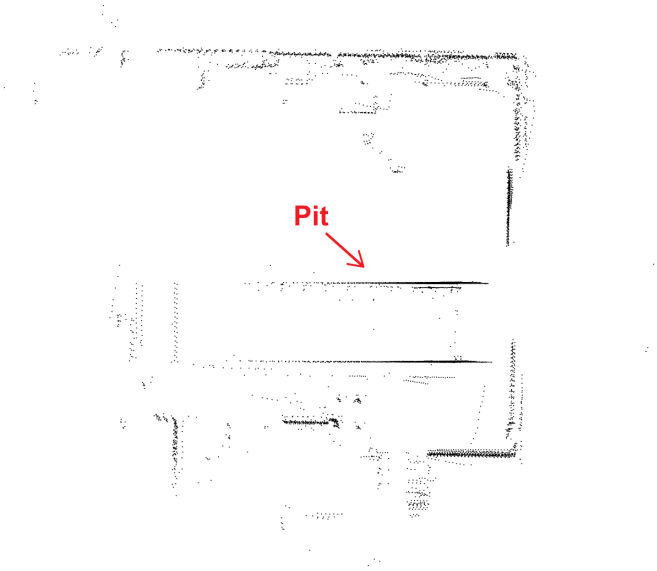
Lastly, in figure 6.17, the walls on the outer edges of the map have a slight rotational creep. This is due to a slight discrepancy in the heading information given by the orientation sensor during the flight, caused by drift. The proposed algorithm does not currently account for rotational drift in-flight, with the aim of improved reliability of the scan-matching even in poor correlation environments. Although not severely impacting the robustness and accuracy of the localisation of the robot in this case it may prove beneficial to perform occasional rotational correction using the point-cloud to avoid this scenario from escalating, if it is encountered.

6.5.3 Entering and Exploring a New Area

The UAV may not always start its flight in the room which it is inspecting, this may be due to several reasons, for instance restrictions on access.



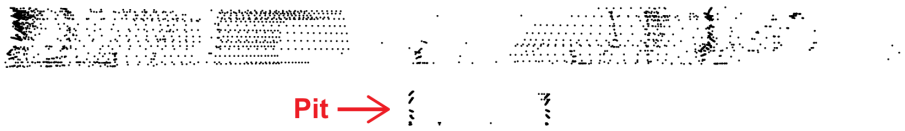
(a) Post-processed high density point-cloud



(b) Real-time low density point-cloud

Figure 6.17: Plan view comparison between the post-processed and real-time maps created from the flight in the pit

(a) Post-processed high density point-cloud

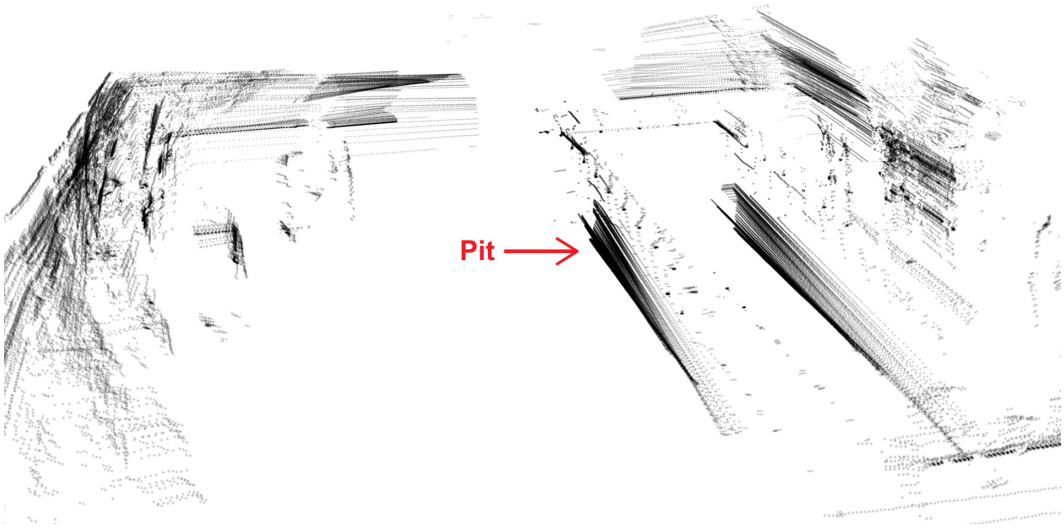


(b) Real-time low density point-cloud

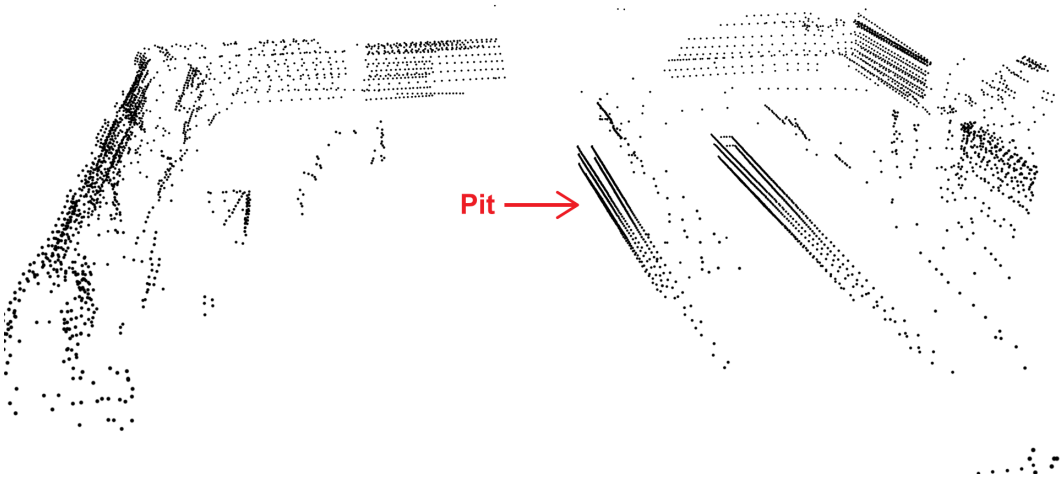
Figure 6.18: Side-view comparison between the post-processed and real-time maps created from the flight in the pit

In this test the UAV was started in a corridor leading to the room, flown around the room and returned to a point near the starting position in the corridor. The resulting point-clouds can be viewed in figures 6.20 and 6.21. The point-cloud shows that the post-processing based SLAM algorithm created a consistent map which joined back at the starting position in the corridor, although with a slight rotational drift.

The real-time low-density navigational map is much the same apart from a slight temporary misalignment, seen on the right of figure 6.20b. It, however, recovered from the misalignment and produced a finishing result similar to the post-processed version, where the initial corridor is aligned after being out of view for a significant portion of the flight, the total flight time being approximately 2 minutes and 10 seconds.

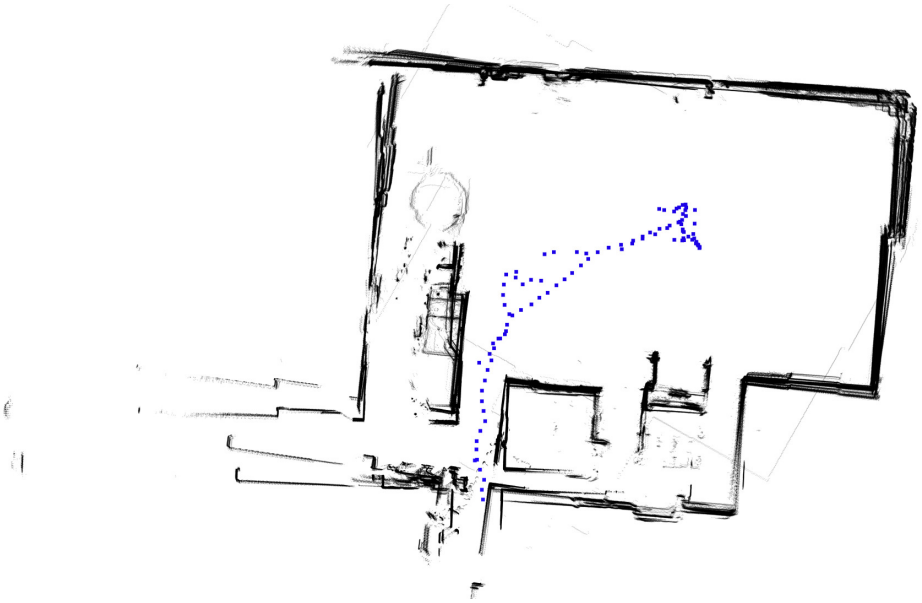


(a) Post-processed high density point-cloud



(b) Real-time low density point-cloud

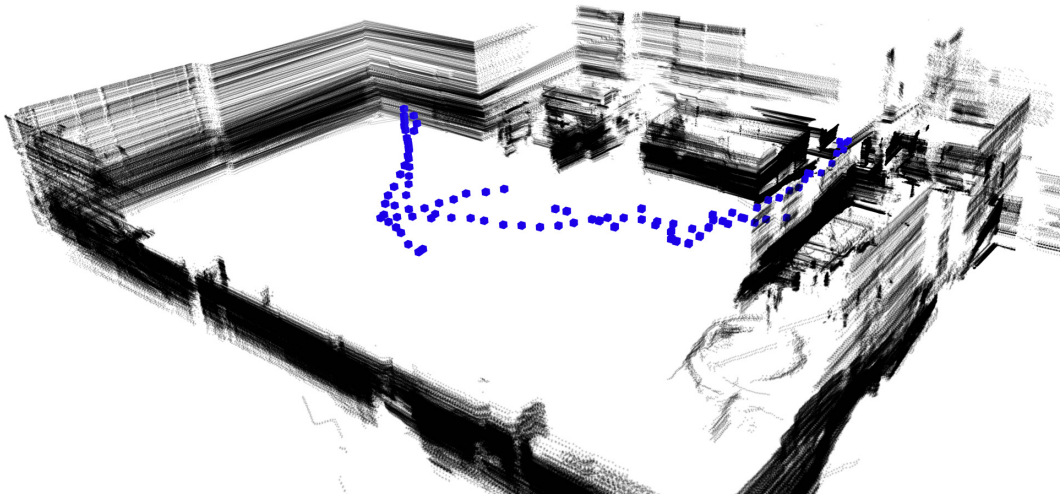
Figure 6.19: Perspective view comparison between the post-processed and real-time maps created from the flight in the pit



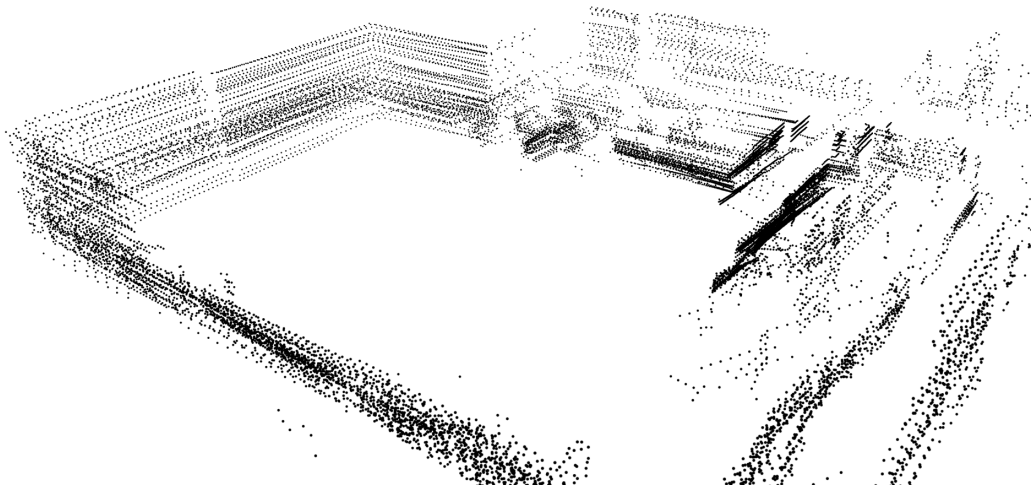
(a) Post-processed high density point-cloud. Including the path of the UAV (blue)

(b) Real-time low density point-cloud

Figure 6.20: Plan view comparison between the post-processed (a) and real-time (b) maps created from the flight in the large hall.



(a) Post-processed high density point-cloud. Including the path of the UAV (blue)



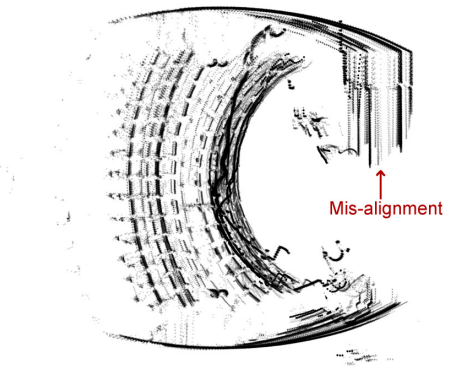
(b) Real-time low density point-cloud

Figure 6.21: Perspective view comparison between the post-processed (a) and real-time (b) maps created from the flight in the large hall.

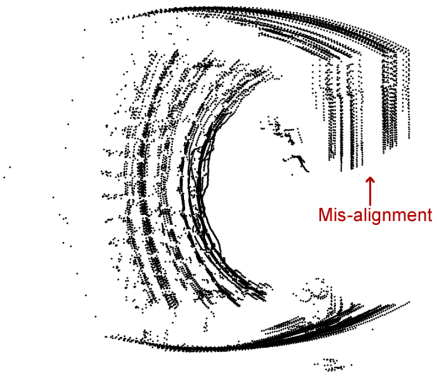
Figure 6.22: The auditorium used to test the performance of the SLAM algorithm in an irregularly shaped room.

6.5.4 Irregularly Shaped Environment

The SLAM algorithm is stipulated to perform poorly in a room comprising solely of sloping walls in the z-axis (height). Flying in a environment with sloping walls would make the UAV see a constantly changing environment as it climbs or pitches, and thus making it difficult for the SLAM algorithm to calculate and maintain an accurate fix of the UAV's position. An environment, which resembles this type of structure, is an auditorium. The room used for this test is shown in figure 6.22.

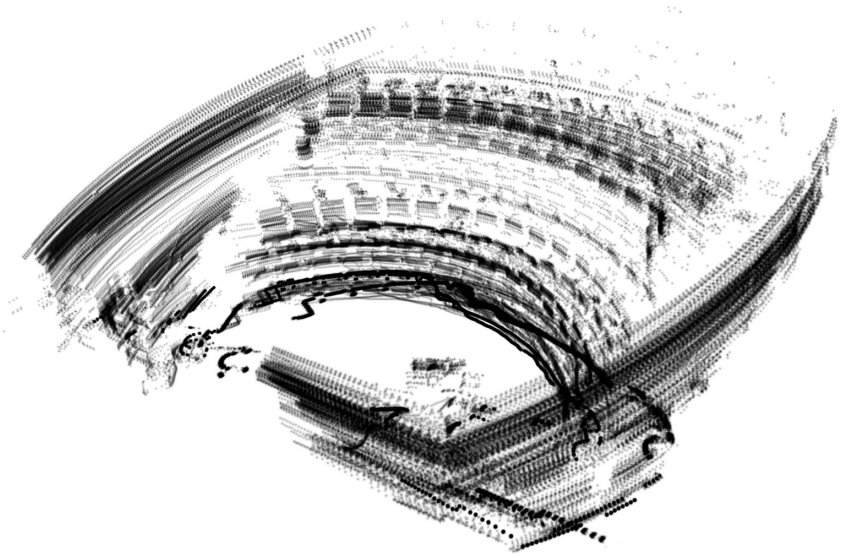


(a) Post-processed high density point-cloud

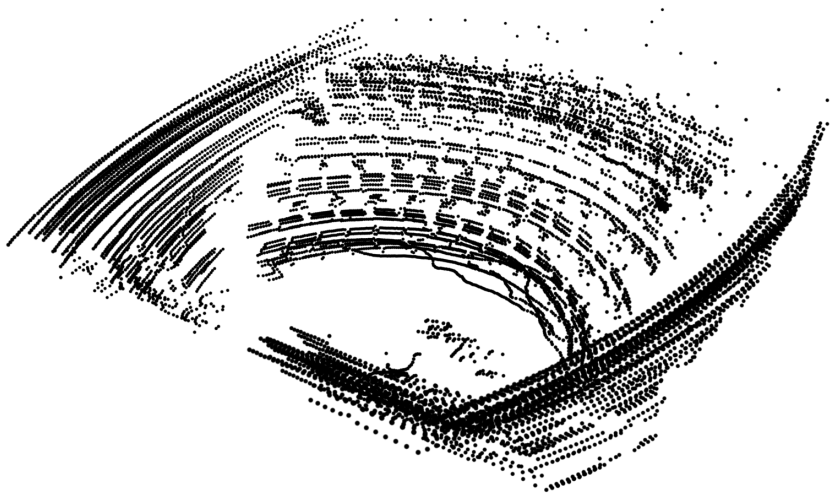


(b) Real-time low density point-cloud

Figure 6.23: Plan view comparison between the post-processed (a) and real-time (b) maps created from the flight in the auditorium.



(a) Post-processed high density point-cloud



(b) Real-time low density point-cloud

Figure 6.24: Perspective view comparison between the post-processed (a) and real-time (b) maps created from the flight in the auditorium.

The UAV was flown facing the seats (manually, without autonomy), thereby emulating the room as an approximate ramp-like structure as much as possible. The resulting maps created by the SLAM algorithm can be viewed in figure 6.23 and

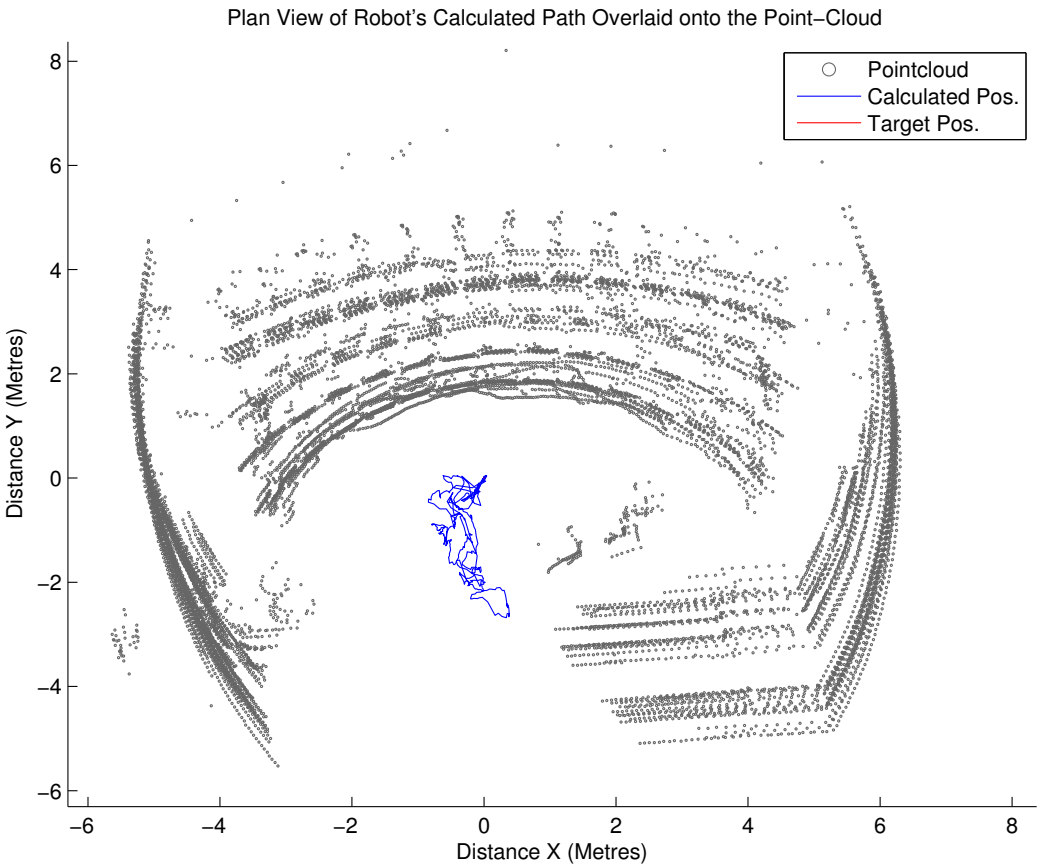


Figure 6.25: The incorrectly calculated path of the UAV overlaid onto the point-cloud

6.24. Although the SLAM algorithm retained an approximate representation of the room, there is significant drift occurring at the upper levels in the y-axis caused by the ambiguity of the structure's layout. The x-axis localisation however remains stable due to the symmetric and consistent nature of the side walls.

Figure 6.25 shows the path calculated by the UAV. The true path of the UAV was a forwards motion (positive y-axis) over the seats before returning to land, the video of which is available in Appendix D.2. Instead the UAV calculated that it had moved backwards (negative y-axis) due to the misaligned position estimates. If this had been an autonomously flown flight, the UAV would have been unable to distinguish that it was calculating an erroneous localisation drift and it would have corrected the position of the UAV to match the perceived drift. If erroneous position estimate

was transitory as it climbed or explored the new area, control would be regained, although the UAV's reference frame to the starting position would be impaired but manageable by the operator, such as the case with the results of this test. However, if the map disintegrates completely, control of the UAV would be lost.

Chapter 7

Flight Testing

Flight testing has been an on-going integral part of the project. Flight tests have been performed in many varying locations to test the UAV system as a whole and to test the performance of individual algorithms through-out development.

This chapter presents the results from a *selection* of these tests, each demonstrating particular conditions or scenarios likely to be encountered during use in its intended application. All the results below, unless stated otherwise, shows the performance of the latest iteration of the UAV design and there is no "tweaking" of the UAV's programming to enhance its performance for each particular test.

7.1 Full System Testing

To test the UAV's compliance with the required objectives set at the start of the project (see Chapter 2 on page 8), a number of *full system* tests were conducted. Although testing the UAV in the chimney would be the ultimate test, this unfortunately was not possible due to the extensive health and safety requirements of testing a prototype robot at a nuclear facility.

Instead a number of tests were designed to demonstrate the proposed UAV's function and performance and was conducted in a manner as if the UAV was used in its intended application.

Figure 7.1: An image showing the operator work station along with the test arena.

7.1.1 Test Methodology

This test was designed to test *all* of the aspects of the UAV's functionality and aimed to demonstrate whether the proposed UAV satisfied the key objectives set at the start of the project. The three key components to this test were as follows:-

Firstly, an operator who had no experience in flying remote controlled helicopters/aircraft was chosen to pilot the UAV for the duration of the mission. The operator was given approximately ten minutes of tuition in how the GUI on the laptop operates, the layout of the functions on the game controller and a brief discussion on detecting and mitigating various failure modes. The operator was placed with his back to the test arena, as shown in figure 7.1, to discourage "peeking" as it was meant to be flown solely through the laptop GUI and the video system. This was to test the ease of use of the UAV.

Secondly, a mission briefing was given to the operator, which depicted how the test should be flown and what to inspect. The diagram given to the operator is shown in figure 7.2. It depicts a flight taking off from the yellow X, flying towards the

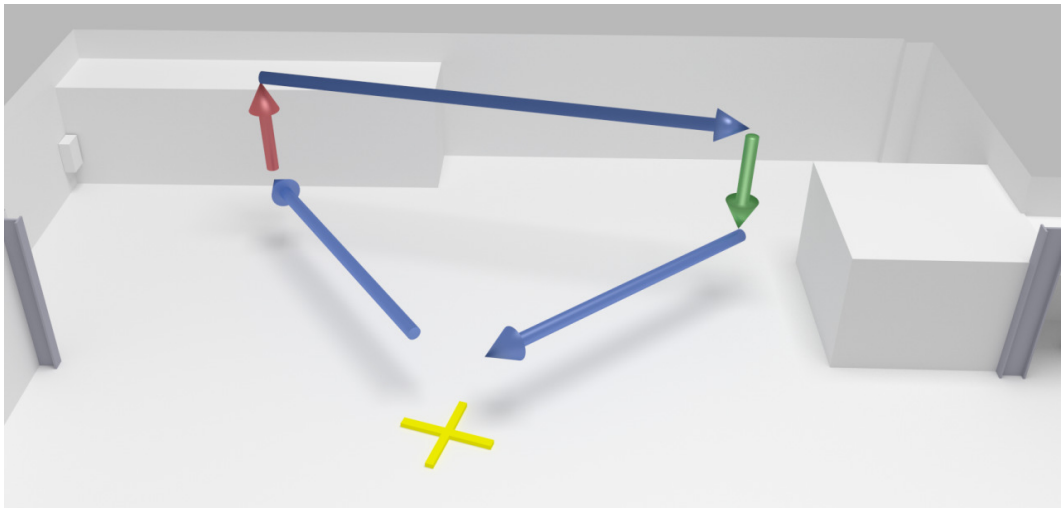


Figure 7.2: Illustration given to the operator, to help visualise the aims of the mission.

ledge on the left, climbing above it later turning and moving over to the ledge on the right, later descending and returning to the X to land. This tested that the operator could fly a required route and that the control and autonomy of the UAV was capable of three dimensional move instructions while remaining in control at all times.

Thirdly, the SLAM algorithm was being tested, although transparent to the operator the mapping algorithm should be able to locate the UAV at all times during the flight, particularly in the specified climb and descend stages of the flight, where the ledges form large step changes in the perceived data as the UAV flies above them, thus testing the robustness of the three dimensional localisation capabilities.

If the UAV's autonomous control was revoked through the use of the safety controller, the flight would be deemed a failure.

7.1.2 Results and Discussion

During the flights all the key functions of the UAV were recorded much like an aircraft's black box. Using this data it was possible to determine how well the various

Flight-ID	Operator	Flight Time	RMSE from target position (m)	Max drift (m)
Full-1	1	6 min 5 s	N/A	N/A
Full-2	2	4 min 17 s	0.90	2.85
Full-3	1	2 min 47 s	1.17	4.14
Full-4	2	2 min 43 s	1.17	4.07
Full-5	2	3 min 27 s	0.76	2.71
Full-6	2	3 min 3 s	0.72	3.66

Table 7.1: Textual results from the test flights

algorithms functioned, while also determining the overall flight characteristics of the UAV.

Six identical test flights were performed, using two operators. Table 7.1 gives a statistical overview of the flights, with the graphics listed in Appendix C on page 208.

Of the six flights, five were completed successfully. The failed flight (*Flight-ID:Full-1*), was the first flight test performed to determine the robustness, accuracy and performance of the UAV as a whole. Ironically this flight was also the first and (to-date) only significant crash the UAV has encountered, and even more ironically due to a fault in the safety controller as opposed to operator or autonomy error. An in-depth discussion of the crash is reported in 7.1.3 on page 153.

Following repairs, the other remaining flights were successfully completed, achieving the full route without the need for intervention between the operator and UAV. One of the flights, *Flight-ID:Full-2* is analysed thoroughly below, the remainder of the flights can be viewed using the same plot types in Appendix C on page 208.

The first plot, figure 7.4 shows a plan view of the UAV's path overlaid with the target path set by the operator. The take-off and landing point positions are both at 0,0m, and the operator approximately followed the guide given in figure 7.2. The "actual" position of the UAV is the calculated position from the SLAM algorithm as no other method was available for determining its true position.

Figure 7.3 gives a breakdown of the UAV's actual and target positions in the three positional axis: X position, Y position and Height. This graph gives a better overview of the characteristics of the control systems of the UAV as opposed to figure 7.4. To help visualise any motion or significant bias in the position error a polar plot was produced of the lateral position deviations and is shown in figure 7.5.

The last set of graphs demonstrates the performance of the SLAM algorithm, with figure 7.6 showing the navigational map created by the UAV as it flew the mission. Figure 7.7, gives an indication of the time taken to process each individual position update, along with the output of the velocity estimator shown in figure 7.8.

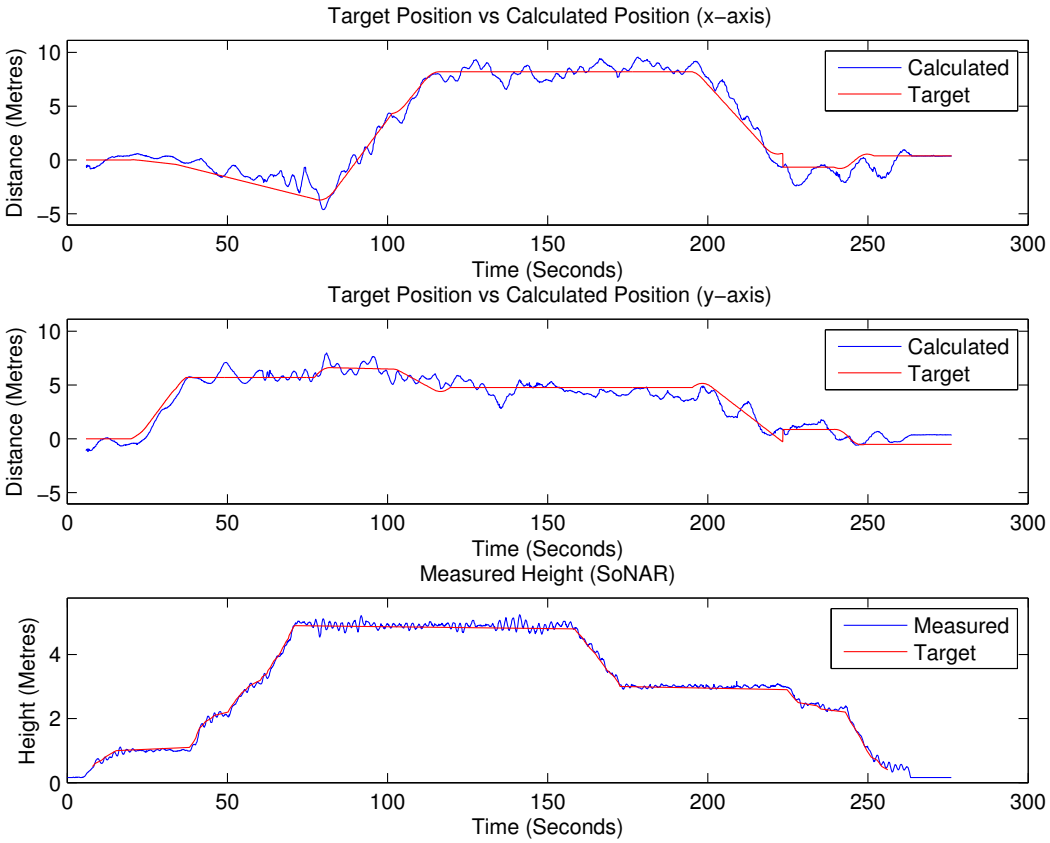
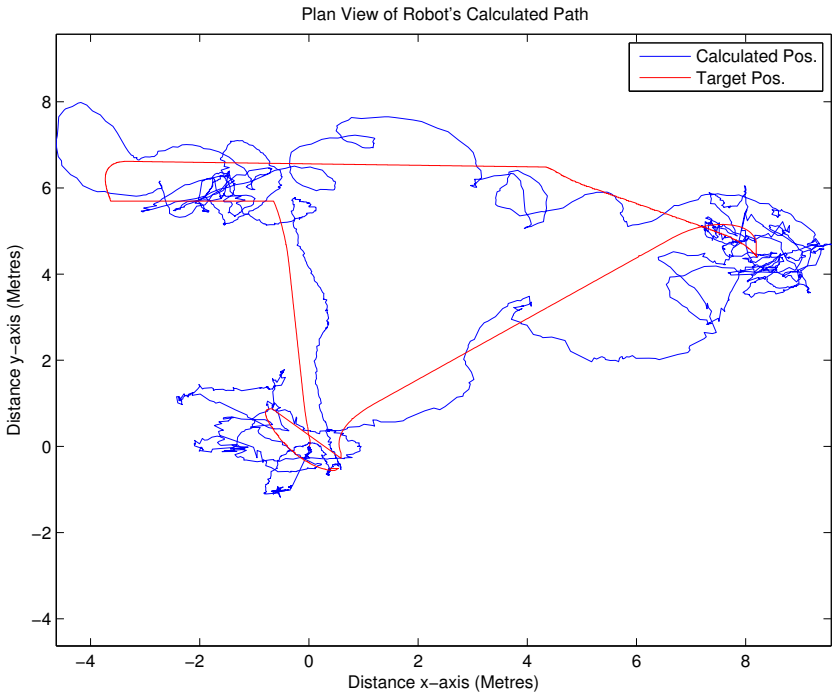
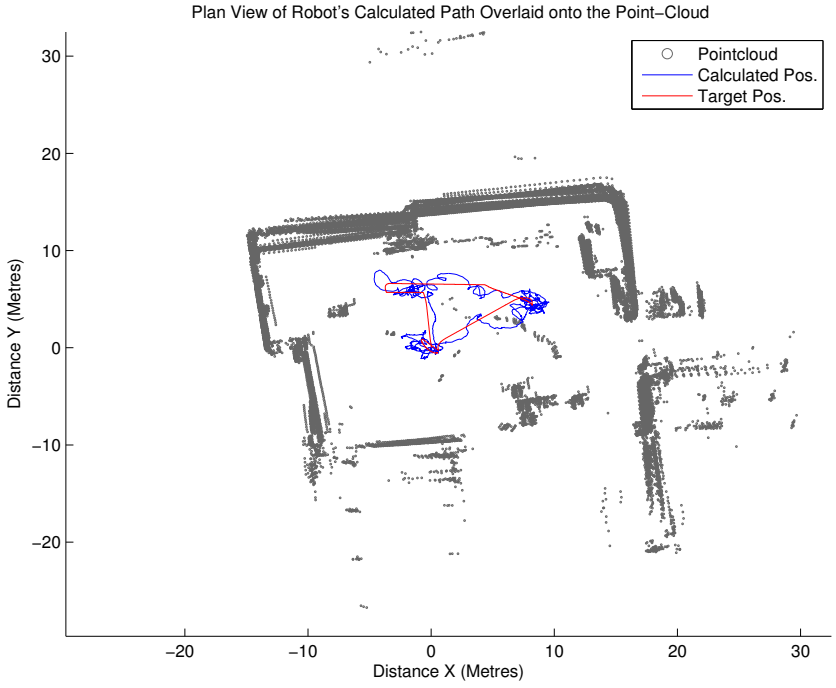


Figure 7.3: UAV's target position and actual position in X,Y and Height axis (*Flight-ID:Full-2)*



(a) The UAV's path



(b) UAV's path overlaid onto the resulting point-cloud data

Figure 7.4: A plan view of the UAV's achieved path and target positions (*Flight-ID:Full-2)*

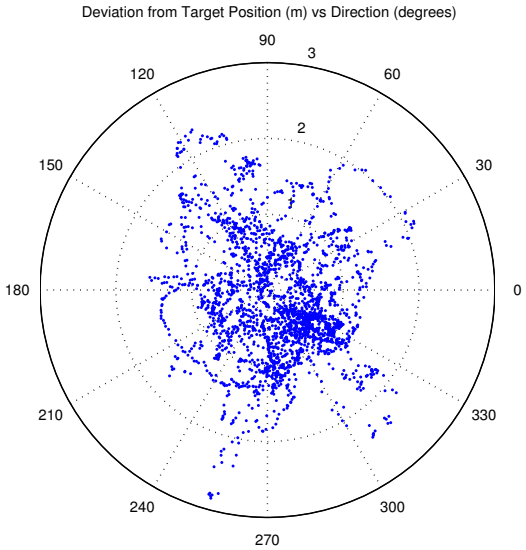
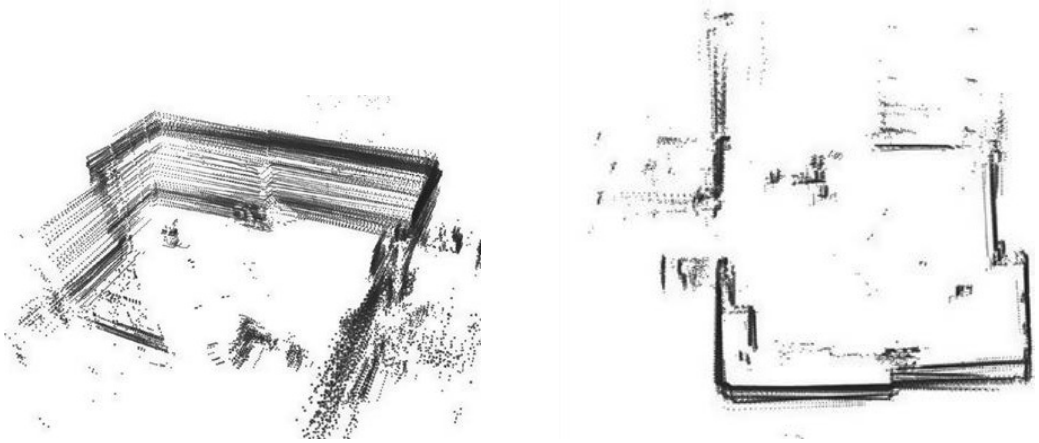


Figure 7.5: Polar plot of the UAV's deviation from the target position (*Flight-ID:Full-2)*



(a) Perspective view (b) Plan view

Figure 7.6: The resulting on-board real-time SLAM navigation map (a), plan view (b)

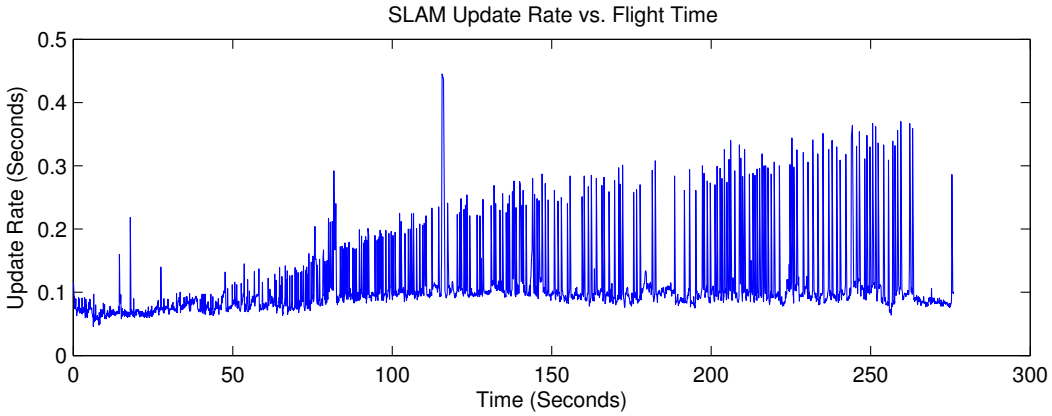


Figure 7.7: Update rate (processing time) of the SLAM algorithm during the flight (*Flight-ID:Full-2)*

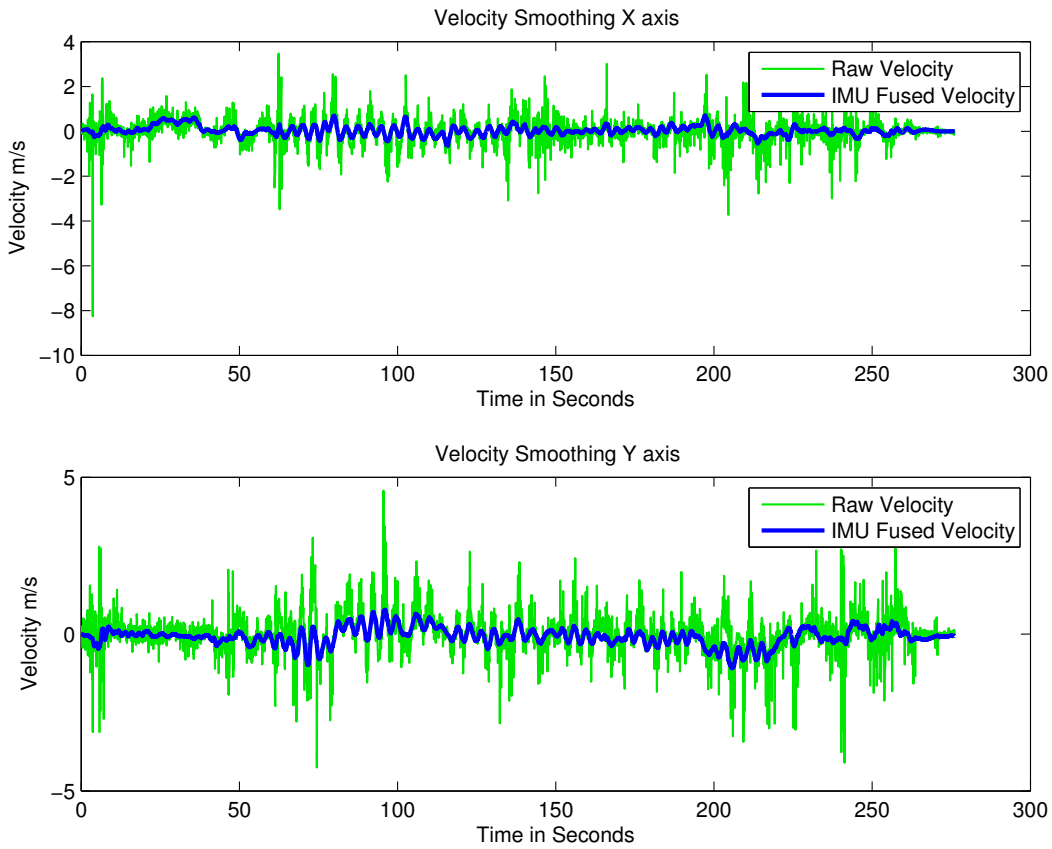


Figure 7.8: Output of the Velocity Estimator through-out the flight (*Flight-ID:Full-2)*

As demonstrated in these figures, the UAV remained aware of its own location through-out the mission, successfully proving the desired function of the mapping algorithm in this environment. Although there were numerous large lateral deviations from the desired track, the PID control loops remained damped and no diverging oscillations developed.

In summary of the six test flights flown, a number of key reoccurring observations regarding the functions of the robot were made:-

Firstly, the tests showed that an operator with little or no knowledge of operating flying vehicles or robots repeatedly completed the set inspection mission successfully and safely, proving the fulfilment of the "ease-of-use" requirement.

Secondly, the SLAM algorithm maintained a robust map of the environment and localised the robot within the generated map without significant error.

Lastly, although the control loops maintained control of the UAV and did not develop divergent oscillations, the precision was far from perfect, with the maximum positional deviations often spanning many metres with a *Root Mean Square Error* (RMSE) of approximate one metre. The control loops, although forming an essential role in this project, have not been the focus of the project and require significant further work before the UAV should be fully deployed.

One reason which causes the large deviations is due to the over-damped PID loops. To prevent divergent oscillations, the PID loops have been tuned to not react aggressively and take a "slow and safe" approach to controlling the UAV. This method perhaps functions well if the UAV is in a hover condition. However, as soon as the UAV is instructed to move or subjected to an external force like a gust of wind, the UAV will be slow to respond and can traverse many metres, potentially in the opposite direction, before again re-aligning with the set-point.

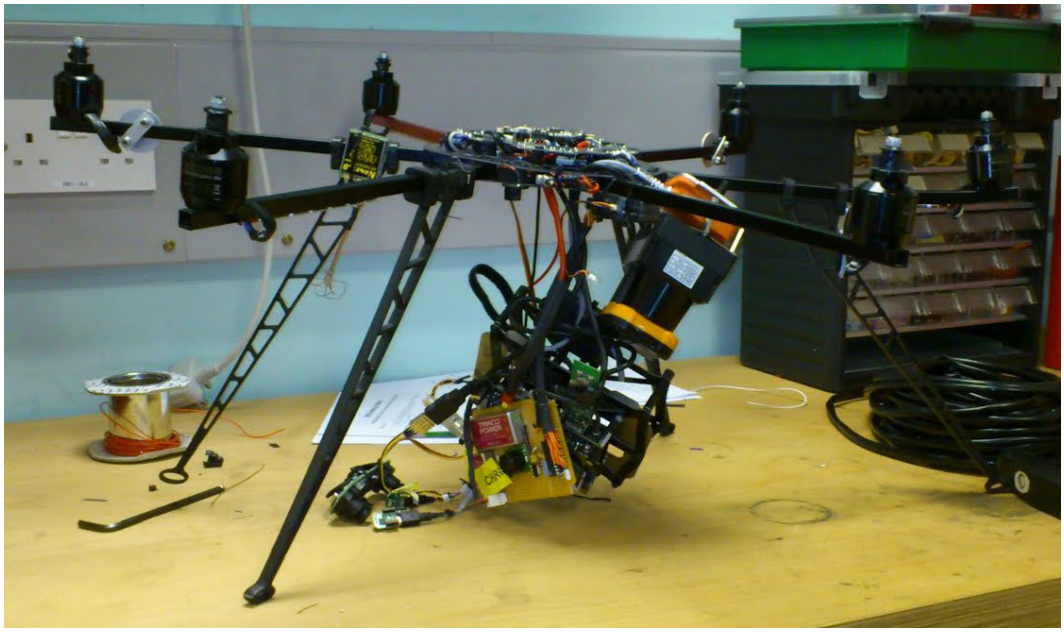


Figure 7.9: UAV after the crash

7.1.3 The Incident (*Flight-ID:Full-1)*

The very first "official" test flight ended in a disaster, with the HexaKopter flying at high-speed into a wall and falling approximately 5 metres causing the extensive damage shown in figure 7.9. All of the outrigger booms were twisted by the impact and all of the rotors shattered, as demonstrated in figure 7.10. Even one of the permanent magnets inside a brush-less motor was cracked, shown in figure 7.11.

Three videos depicting the crash are included in Appendix D.1, two from cameras recording the flight and one of the operators GUI. The videos depict the UAV flying autonomously as planned, then suddenly establishing a slight climb along with a forwards motion. A recovery was attempted using the safety controller, however there was no response. Even after activating the emergency stop, the UAV simply continued on its path into the wall in a runaway state, with no change in attitude.

Figure 7.10: One of the bent and twisted outriggers.

Figure 7.11: The rotor of one of the brush-less motors, showing a shattered permanent magnet

7.1.3.1 The Cause

The force of the crash caused the on-board computer to suddenly reboot, resulting in the corruption of the log files. The only method of determining the cause of the crash was to analyse the footage and the video recording the operator's GUI.

Firstly, it was determined that the on-board computer was responsive and in full working order at the time of impact. All the sensors appeared to be functioning, also a collision avoidance warning was given as the UAV impacted the wall. The PID loops were shown to be functional, outputting commands to the UAV in the correct sense.

Even if the computer were unresponsive it cannot degrade the functionality of the safety controller by design (see 5.5.5 on page 113). The safety controller survived the crash *almost* intact and was easily repaired. After a number of hours using an oscilloscope to attempt to recreate the lockup state, a particular condition was found which mimicked the situation which caused the crash.

If there was a momentary glitch between the safety controller and the USB-Serial converter, the safety controller would become unresponsive to any external inputs and would, however, continue to output the last valid control signal to the HexaK-opter. It would only recover upon cycling of the power supply. It was discovered that the serial input pin had been left floating (without a pull-down resistor), thus when the serial converter was removed, the noise on the wire caused a large amount of interrupts. Causing the safety controller to loose its syncronisation with the RC receiver.

The safety controller had been programmed to automatically perform an emergency-stop halting the motors in this situation, however a "bug" in the software prevented the correct functioning of this feature in this particular scenario.

7.1.3.2 The Damage and Repair

On the Hexakopter platform all of the outrigger booms had twisted and two motors were damaged beyond repair. Most of the UAV was built using plastic bolts, many of which sheared. All the motor controllers remained functional, however due to a short circuit the HexaKopter flight controller became unresponsive and was damaged beyond repair.

The three-dimensional stereoscopic camera previously used for the operator was also broken along with the Lithium Polymer battery, which although still functional was discarded due to impact damage. Thankfully all of the sensors, including the computer survived intact with no apparent damage.

7.2 Other Notable Test Flights

In order to fully evaluate the performance of the UAV it needs to be flown in many different environments. This is especially important in this project as it focuses on the development of the SLAM algorithm. It is important to ensure that it functions not only in the environment it was tested during development, but also many other environments with varying geometry and features.

7.2.1 Sellafield Demonstration

During the latter stages of the project a demonstration to Sellafield's employees was organised, partly to demonstrate the progress and capabilities of such a UAV but also to test the UAV's function in a new yet possibly similar environment to which it may be deployed.

Figure 7.12 shows the building where the UAV was flown. Although relatively homogeneous in height, there was a lot of clutter which could cause erroneous readings.

(a) View towards the operator station

(b) View from the operator station (where the UAV was flown)

Figure 7.12: The building in which the Sellafield demonstrations took place

During the demonstrations numerous flights were used to collect sample data to which a highly detailed post-processed map could be created to demonstrate the potential accuracy and detail of the generated maps. The resulting map from one of these flights can be viewed in figure 7.13.

Also, a few autonomy demonstrations were conducted to show that a competent pilot was not needed to fly the UAV. This was performed by allowing members of the audience to have an attempt at taking off, moving down the hall, then returning to land.

Data from one of these flights is shown in figures 7.16 – 7.15 and the resulting internally produced maps are shown in figure 7.14. During testing the point and click navigation method was still experimental and was discouraged, instead the operators were told to use velocity control. When using velocity control, the operator controls the UAV's target velocity (using the game controller) instead of its position, although when zero velocity is requested the UAV re-enables the positional control to maintain a steady hover.

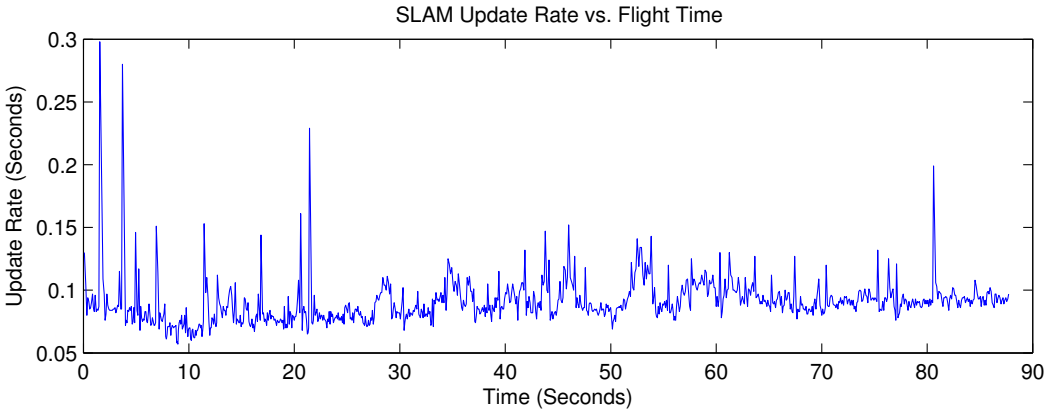
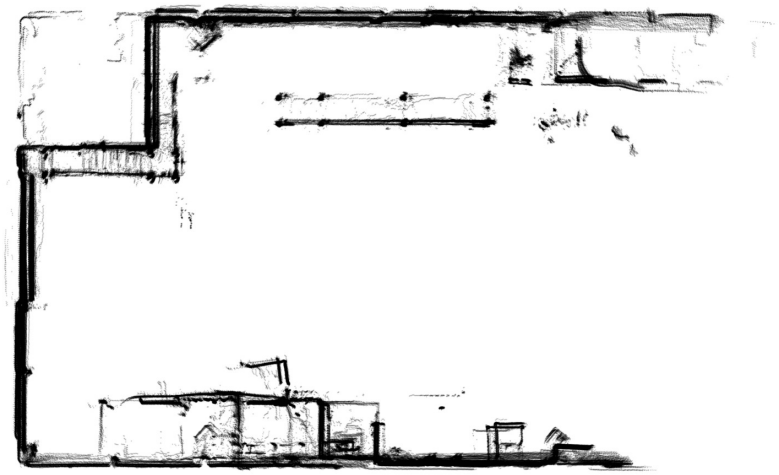
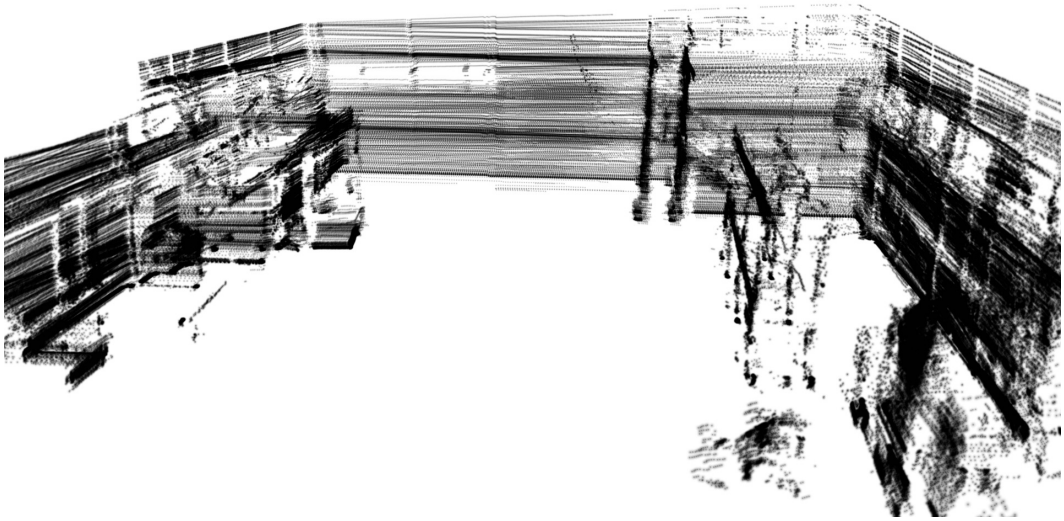


Figure 7.15: The SLAM algorithm refresh rate during the flight

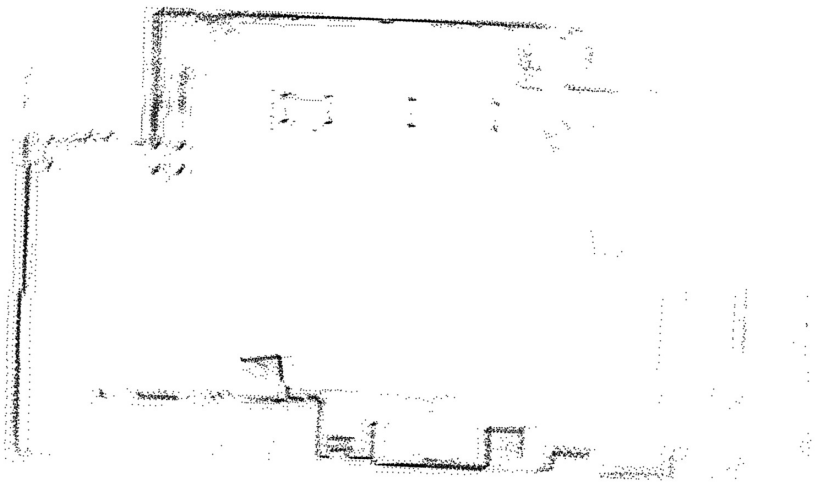


(a) Plan View

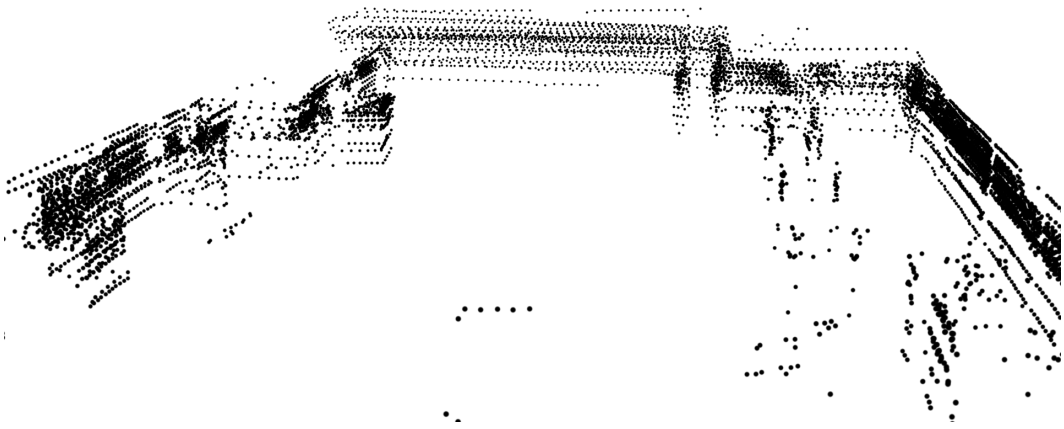


(b) Perspective View

Figure 7.13: High density, post processed map of the Sellafield demonstration area



(a) Top View



(b) Perspective View

Figure 7.14: Low density, real-time navigational map of the Sellafield demonstration area

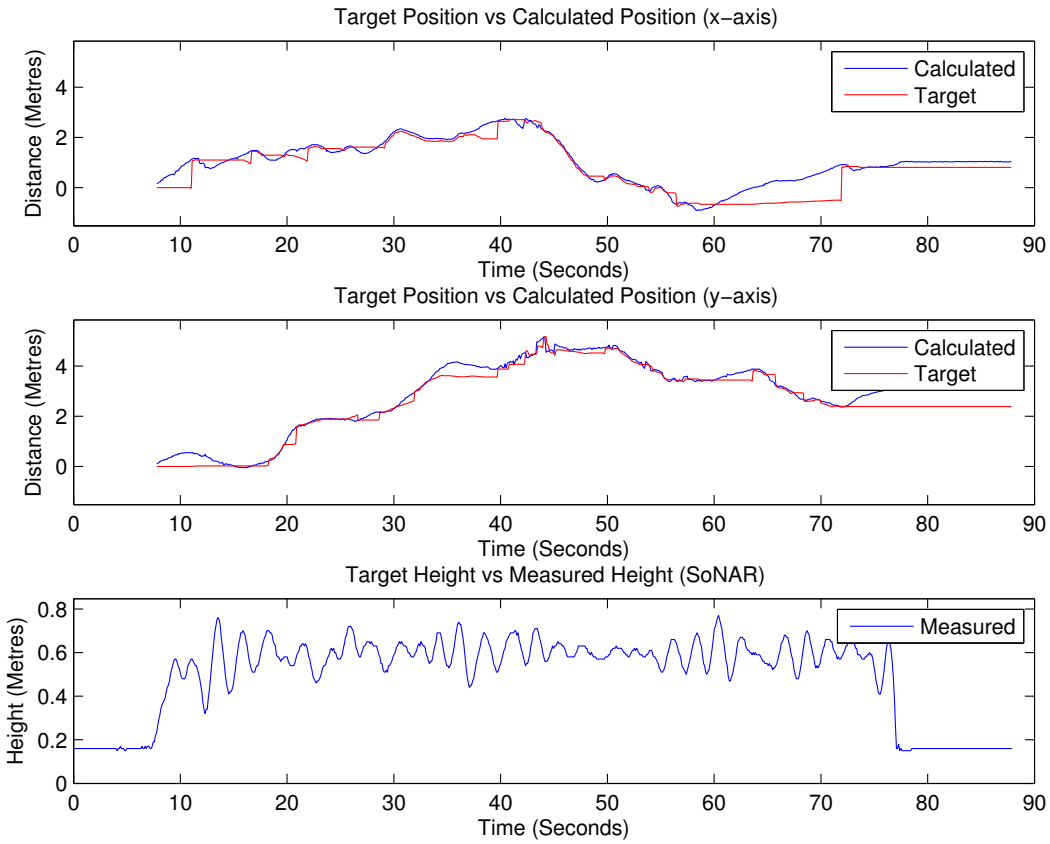
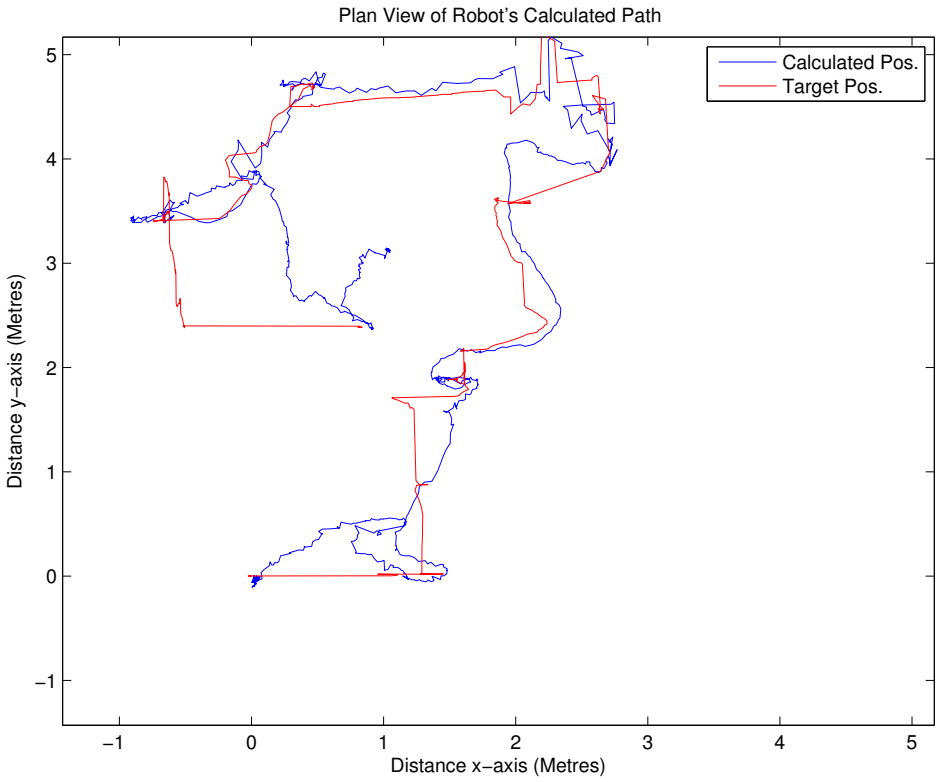
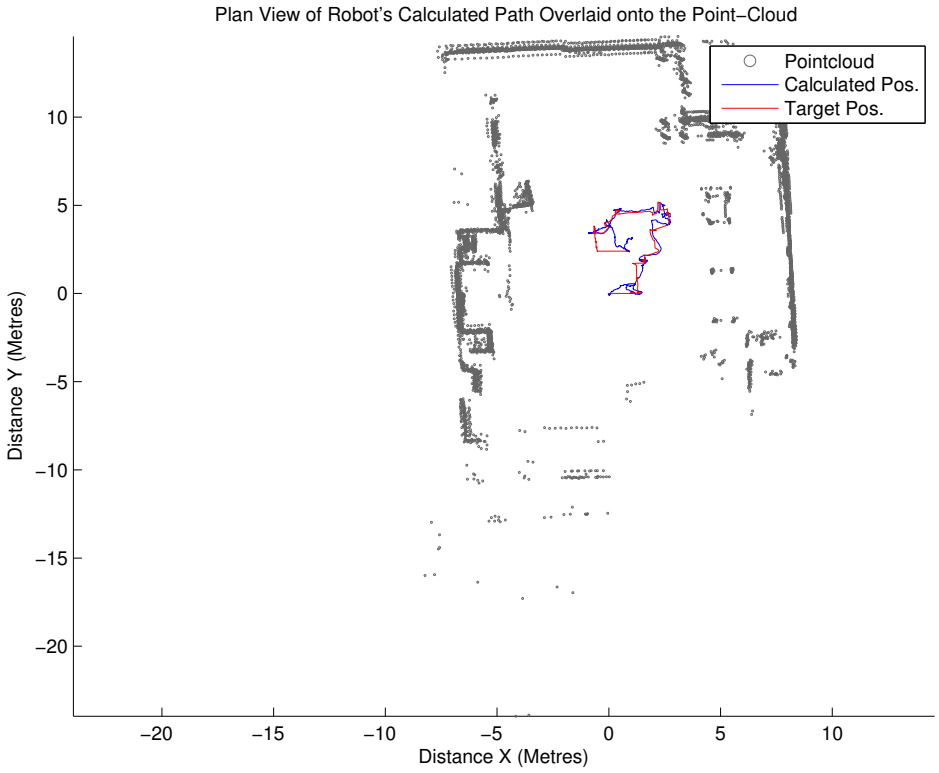


Figure 7.16: Target vs actual position for the “autonomous” flight



(a) Without point-cloud overlay



(b) With point-cloud overlay

Figure 7.17: Plan view of the robot's path

Figure 7.18: Gantry used for the close quarter flying test

7.2.2 Close Quarter Flying

A notable flight performed during the development was a short flight through a confined area, designed to evaluate and demonstrate the control available to the operator. The operator had little experience of flying the UAV, had line-of-sight view of the UAV, although often obscured, as well as access to the UAV interface.

The test involved the gantry shown in figure 7.18, whereby the UAV was flown into the structure between the pillars, shown in figure 7.19, along the structure and out through the final pillar at the end. Again, the point and click navigation method was still experimental and was discouraged, instead the operator relied on utilising the velocity control feature. This meant that the UAV was flown using one of the analogue sticks on the game controller in order to control its velocity throughout the mission.

Figure 7.19: UAV flying through the pillars in the gantry

During the flight a number of key observations were made, a video of which is available in Appendix D.1:-

Firstly, after exiting the structure at the very end of the flight a bug which has subsequently been fixed in the motion planner erroneously reset the position of the set-point of the PID loops to the take off point. This can be seen clearly in figure 7.20 and 7.21 and caused the UAV to loose control and land heavily, luckily with no damage.

Secondly, the SLAM algorithm maintained a fix throughout the flight, reliably reporting the UAV's position, even when passing objects and exploring new features as it entered the structure. Unfortunately, the navigational map was not logged for this flight.

Thirdly, the precise control of the UAV's path can be achieved through the velocity control feature, although it requires the operator to pro-actively guide the UAV as when engaged the positional PID loops are disabled. If no velocity instructions on a particular axis is given the UAV reverts to hover mode with the positional PID being active, such as shown in figure 7.20 on the x-axis graph around 200s.

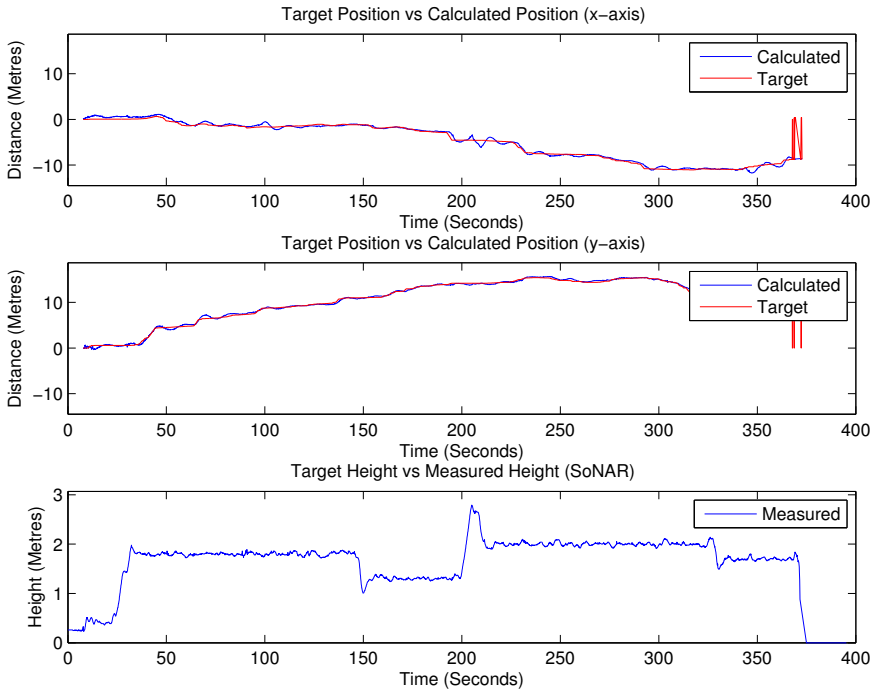


Figure 7.20: A graph showing the actual and target position of the UAV during the flight.

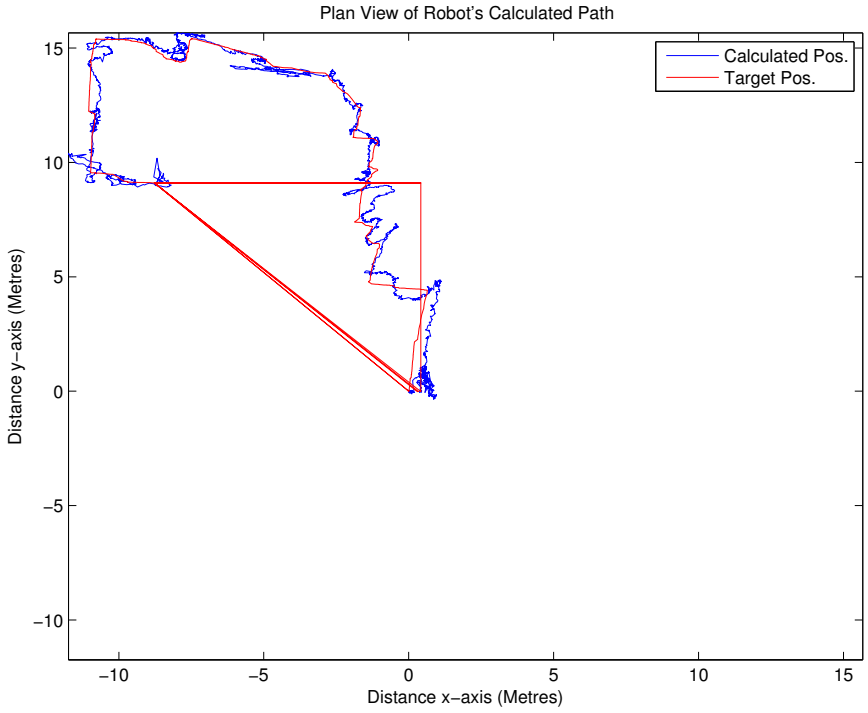


Figure 7.21: A top down view of the UAV's path through the structure.

Figure 7.22: Climbing and descending into obstacles are a potential collision risk

Lastly, although the UAV has lateral collision avoidance, climbing or descending into a collision scenario is still possible with the current sensor layout. Such a situation was encountered during this flight and is depicted in figure 7.22, however in this case it was noticed and avoided by the operator.

7.3 Summary and Discussion

This section critically discusses the results and compares them to the requirements set at the outset of the project, discussed in Chapter 2 on page 8.

7.3.1 Operating Environment

As stated in 2.1 on page 8, the UAV should be able to operate inside a 125 metre tall, 15 metre diameter chimney with clutter and, potentially, hanging cables.

Unfortunately, gaining access to test the UAV in the chimney has not been possible at the time of writing, primarily due to the lengthy approval and risk assessment

process needed to enable a prototype robot to enter an active nuclear facility. Instead the UAV has been flown in several other large environments to test its capability.

Climbing to 125 metres and returning to ground-level is within the capability of the UAV platform, discussed further in 7.3.3. This would far exceed the SoNAR sensor's approximate seven metre operating range, however, the altimeter should be capable of providing sufficiently detailed height information to complete the mission as shown in 6.1.1 on page 115. Although, if higher accuracy height information is required, then another method of registering the UAV's height needs to be investigated.

The hanging cables are a problem with the current UAV design. The LiDAR scanner is able to detect a standard 13amp cable from approximately 1.5 metres distance. Due to the less than perfect control of the UAV's PID loops attempting to avoid a full speed (0.5m/s as stated in 5.3.5 on page 86) collision in less than 1.5 metres is currently not a possibility. As discussed further in 7.3.3 it may be beneficial to use a rotor guard to lessen the risk of damage during small impacts.

7.3.2 Inspection

As stated in 2.2 on page 9, the UAV should be able to record imagery of the building, measure the levels of radiation and also be capable of producing three-dimensional point-clouds to aid the task of inspection.

The UAV has an on-board camera capable of recording both high resolution stills and video. Although the camera performs badly in low light it is improved marginally by the basic on-board lighting system. The camera, however, is not an integral part of the UAV platform and can easily be replaced with other light-weight camera systems, such as an infra-red camera for low light conditions.

The UAV is currently unable to detect radiation, as no radiation sensor is currently mounted to it. There are many light-weight dosimeters available on the market which could easily be attached and integrated into the system.

Three-dimensional point-clouds can be produced by the UAV's sensors. The point-clouds can either be generated using the proposed algorithm however without the density limitations, as explained in 5.4.3.3 on page 103, or through the use of commercial point-cloud registration software. It was not within the scope of the project to create high-accuracy post-processed point-clouds, instead the focus was on building and developing the UAV. Many tools already exist for point-cloud manipulation, common within the GeoInformatics field.

7.3.3 Robotic Platform

Linking back to the requirements regarding the robotics platform in section 2.3 on page 10:-

7.3.3.1 Compact Robot

The robot needs to be compact, ideally being able to be inserted into the operating environment through inspection holes. This requirement is not achievable or attempted with the proposed UAV design, it should however be achievable in future iterations of the design, by for instance implementing a foldable airframe, such as the Draganflyer UAV pictured in figure 7.23.

7.3.3.2 Sufficient Flight Time for an Inspection Flight

The flight time of the proposed UAV is approximately *twelve* minutes, given a hover at around $3-5$ metres. Twelve minutes should be sufficient to monitor a number of areas, as demonstrated in the flight-tests in chapter 7 on page 144. For extensive inspection missions it will be necessary to survey part of the building, land and recharge the UAV or replace the battery to later continue the mission.

Autonomous "high" altitude tests have not yet been performed, however the UAV will require a modification to the control theory to enable rapid climb/decent mode

Figure 7.23: The Draganflyer X4-P, featuring a foldable airframe [66]

to reduce the flight time as much as possible, as for these mission the climb and descend time can become a substantial part of the overall flight time. A disadvantage of the multi-copters is that there is no auto-rotation feature as with helicopters, meaning a loss of motor power equates to loss of control of the craft.

7.3.3.3 Airborne and Non-Contact

The HexaKopter platform is by design a self-sufficient airborne system and does not require any physical contact or contact guidance through-out the mission.

Contact will however occur if it collides with an object during the flight. At present the rotors are the first point of contact, hence they will break the rotor and may cause the UAV to lose control. To mitigate this it may be beneficial to fit the next iteration of the proposed UAV design with a rotor guard, which will protect the rotors from slow speed collisions or hanging wires, however, it will add weight and lower the overall flight time.

7.3.3.4 Sufficient Payload for Sensors

There is always a compromise between the weight of the UAV and its effective flight time and through weight reduction it may be possible to extend the UAV's flight time. The sensors and devices mounted on the UAV weigh combined approximately 800g (see 4.4 on page 65) excluding battery, cabling and connectors. It should be possible for the UAV to effectively lift an extra 100 – 200g without serious impact on flight performance thus allowing sensors or cameras to be upgraded to larger, heavier products.

7.3.3.5 Cost

The airborne part of the UAV is required to cost less than £10,000. The cost of the proposed system is approximately £7,200 (see 5.2.3 on page 78) which satisfies this objective.

7.3.4 Autonomy

Ease of use was a primary requirement for this project, with an emphasis on a minimally-trained operator being able to safely use the basic functions of the UAV. This was successfully demonstrated during all the flight-tests in Chapter 7 on page 144. During the tests an unskilled operator was used and successfully demonstrated that the UAV could be flown with little to no training, giving the UAV waypoints on where to fly as opposed to physically controlling it.

The proposed UAV is however not without its flaws. As demonstrated throughout the flight tests the control theory requires further work, as the UAV often drifted from its target position by several metres. The problem with control has to some extent been solved in related UAV projects, however interpreting and implementing these control methods require substantial effort and was deemed best suited as further work.

Regardless of the performance of the control theory, without knowledge of where the UAV is and where it should go, control is impossible. It is this aspect which has been the focus of this project, determining the location of the UAV throughout the flight using only the on-board sensors and computer. The tests performed on the proposed SLAM algorithm has demonstrated it to be capable and reliable in the sample environments analogous to the environments it is to be flown. In certain conditions the algorithm is known to give erroneous results, such as where the majority of the walls in the room change asymmetrically with height, as shown in the test in section 6.5.4 on page 139.

Due to the simplicity of the algorithm only single rooms can be reliably explored. The proposed SLAM algorithm does not use loop-closing or other statistical methods to improve the accuracy of the map, instead its focus is on speed and robustness within a single enclosed environment.

The decision to completely remove the rotation estimator from the SLAM algorithm has introduced both negative and positive aspects. The positive aspects are that the algorithm requires less information to scan-match each scan as one of the unknowns, the rotation, is already solved, meaning that an accurate match can be found even in a cluttered or obscured environment, where the original algorithm would fail. The negative is that the maps produced are prone to rotational drift, where the orientation sensor is outputting a slightly incorrect heading. This drift is slow, however accumulative. The best solution is perhaps a compromise, running the rotational estimator once every 10 seconds to re-align the point-cloud to reduce the onset of these errors.

Chapter 8

Further work

The following is recommended as further work for the successful development and deployment of the UAV for use in industry:-

Control Theory: The positional PID control structure needs to be revised or a different control method used. If the dynamics of the system can be analysed so that an approximate mathematical model of the UAV can be created, then a model or predication based control system can be implemented allowing for more accurate and reliable control.

Simulation: Accurate simulation of the UAV's dynamics and sensors would enable preliminary testing of features and algorithms before actual flight-tests are performed. This would improve safety as the algorithm or function can be fully debugged and checked to be working in the correct sense before flight. It would also improve the reliability as various failure modes or difficult environments can be emulated and tested without risk.

Further Miniaturisation: Computer and sensor technology is a rapidly progressing field. If the proposed UAV is to be developed further the new state of the art devices should be considered, as it may allow for a smaller UAV platform to be used, or enable significant improvement in flight-time through a reduced payload. An example of which is the latest-generation smart-phones, they now offer more processing power than the computer used on-board the UAV, are also lighter and

use less power and newer sensor technologies enable higher resolution, lower cost and are both smaller and lighter.

With the development of more powerful computers which are suitable to be used on-board, more advanced control, autonomy and localisation algorithms can be used to counter the limitations discussed in the flight testing *Summary and Discussion* section (7.3) and the SLAM *Limitations* section (5.4.3.2).

Currently a two-dimensional LiDAR scanner is used, relying on accurate height, roll and pitch data to project the scanner two-dimensional plane into three dimensions (visualised in Appendix D.2). Although not yet commercially available, a suitably small and low-weight three-dimensional LiDAR scanner may improve the robustness and accuracy of the SLAM algorithm.

Chapter 9

Conclusions

This project involved the development of an *Unmanned Aerial Vehicle* (UAV) for the purpose of inspecting the internals of buildings or structures, where an operator cannot enter easily or entry is prohibited. At the start of the project a number of requirements were set by Sellafield, the project's industrial sponsor, based on what they would expect from an ideal end product. To ensure that these requirements were met as far as possible, a systems style approach was adopted to help guide the project and was also used to structure this thesis.

A UAV was developed using primarily commercial-off-the-shelf components in an effort to reduce development time and to increase reliability. A six-rotor HexaKopter was chosen as the robotic platform, which was modified to carry the sensors and devices needed to enable autonomy and basic inspection.

A semi-autonomous control approach was chosen, whereby an un-skilled operator could be used to control the UAV, instructing it where to go as opposed to flying it manually. Preliminary tests show that the UAV can be safely flown by an inexperienced operator given a ten minute tutorial on the UAV's various functions.

The proposed UAV's primary weakness is, however, its lack of control. Due to the lack of an accurate simulation environment, significant amounts of testing have been performed throughout the project in an attempt to experimentally tune and improve the UAV's positional PID loops. A frequently encountered problem was divergent oscillatory behaviour, as a result the developed control system is over-damped, resulting in generally convergent oscillations. However, this has the side

effect of non-aggressive control causing large uncommanded deviational drift (greater than 2m).

9.1 The Proposed SLAM Algorithm

After reviewing the current state of the art it was realised that the localisation algorithms used by related research groups were not suitable for this project. A detailed analysis of their solutions revealed that, in order to reduce the SLAM's computational overheads, they resorted to methods such as assuming a certain geometric structure, that the building can be represented in two-dimensions, rely on external (off-board) processing or function on the premise that an accurate model of the building, in its current state, already exists. None of these methods can be assumed in order to fulfil the requirements of this project.

Localisation is a fundamental step in enabling accurate control and autonomous behaviour. Developing a reliable, high-speed, three-dimensional SLAM algorithm, that could function within a computationally constrained UAV, became the primary focus of this research project.

The proposed SLAM algorithm is *novel* in its implementation and is based on the popular *Iterative Closest Point* (ICP) algorithm with a number of modifications to lower the processing requirements, enabling it to run solely on the UAV's on-board computer. These modifications include:-

1. Utilising FLANN[78] as an approximate nearest neighbour search algorithm in an attempt to reduce the time to pair each point with the built "navigational" point-cloud.

2. Reducing the density of the data collected and stored from the LiDAR scanner. Experimentally it was determined that every fifth point (216 total) could be used from each scan without significantly impacting accuracy or the robustness

of the resulting map. Similarly, through selectively adding these scans to the navigational point-cloud, the overall data density and thus processing time could be reduced further.

3. Removal of the *rotation estimation* and *error-function* features from the ICP algorithm. Heading data is instead obtained from the XSens orientation sensor, which increases the robustness of the scan-matching and lowers processing time. However, solely relying on data obtained from the orientation sensor can lead to slight heading drift over time of a few degrees per minute.

Real-world flight testing has demonstrated that the proposed SLAM algorithm is both capable and reliable in environments analogous to the ones where the UAV is expected to be flown. However, in environments comprised of walls sloping with height, the algorithm will most likely return erroneous results unless an accurate model of the building can be obtained prior to the flight.

References

[1] ND Baldwin. *Remediating Sellafield-A New Focus for the Site*. United States. Department of Energy. Office of Scientific and Technical Information, 2003.

[2] L. Arnold. *Windscale 1957: anatomy of a nuclear accident*. Macmillan, 1995.

[3] Sellafield Ltd. http://www.sellafieldsites.com/wp-content/uploads/2012/08/Aerial of Sellafield Site Sept09 349G9437.jpg [Accessed: Nov 2012].

[4] C. Bayliss and K. Langley. *Nuclear Decommissioning, Waste Management, and Environmental Site Remediation*. Elsevier Science, 2003.

[5] PR Lutwyche and SF Challinor. *Sellafield Decommissioning Programme-Update and Lessons Learned*. United States. Department of Energy. Office of Scientific and Technical Information, 2003.

[6] BBC. http://news.bbc.co.uk/1/hi/england/2643063.stm [Accessed: Mar 2013].

[7] S. Kawatsuma, M. Fukushima, and T. Okada. Emergency response by robots to fukushima-daiichi accident: summary and lessons learned. *Industrial Robot: An International Journal*, 39(5):428–435, 2012.

[8] G.A. Khoury. *Airship technology*, volume 10. Cambridge University Press, 2012.

[9] Northrop Grumman. http://www.northropgrumman.com/Photos/pgL_BS-50001_014.jpg [Accessed May 2013].

[10] J.G. Leishman. *Principles of Helicopter Aerodynamics*. Cambridge Aerospace. Cambridge University Press, 2006.

[11] P. Dmitriy. http://www.airliners.net/photo/Russia—Air/Kamov-Ka-50/0920728/L/ [Accessed: Mar 2013].

[12] Gabriel M Hoffmann, Haomiao Huang, Steven L Waslander, and Claire J Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proc. of the AIAA Guidance, Navigation, and Control Conference*, pages 1–20, 2007.

[13] K Sevcik and P Oh. Designing aerial robot sensor suites to account for obscurants. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 1582 –1587, 29 2007-nov. 2 2007.

[14] Mesa Imaging. *SR4000 User Manual*.

[15] Microsoft. http://www.microsoft.com/en-us/kinectforwindows/develop/tutorials/tutorialsdesc. [Accessed Feb 2013].

[16] J. Flatley. http://www.engadget.com/2010/11/08/visualized-kinect-night-vision-lots-and-lots-and-lots-of-do/ [Accessed Feb 2013].

[17] Sparkfun-Electronics. https://www.sparkfun.com/ [Accessed: Jan 2013].

[18] R. Bogue. The fast-moving world of mems technology. *Assembly Automation*, 29(4):313–320, 2009.

[19] US NOAA and U.S. Air Force. Us standard atmosphere, 1976. *Washington, DC*, page 206, 1976.

[20] P. Godwin. *Air Pilot's Manual: The Aeroplane - Technical*. Air Pilot's Manual Series. Air Pilot, 2003.

[21] Freescale. *MPX4115 - Freescale Semiconductor (Datasheet)*.

[22] M F L'Annunziata. *Handbook of Radioactivity Analysis*. Elsevier Science, 3 edition, 2012.

[23] T.J. Prescott, M.J. Pearson, B. Mitchinson, J.C.W. Sullivan, and A.G. Pipe. Whisking with robots. *Robotics & Automation Magazine, IEEE*, 16(3):42–50, 2009.

[24] B. Hofmann-Wellenhof, H. Lichtenegger, and E. Wasle. *GNSS–Global Navigation Satellite Systems: GPS, GLONASS, Galileo & more*. Springer, 2007.

[25] Trimble. *Trimble Pro Series Datasheet http://trl.trimble.com/docushare/dsweb/Get/Document-608950/022501-289C_Pro[Accessed: March 2013]*, 2012.

[26] Youjing Cui and Shuzhi Sam Ge. Autonomous vehicle positioning with gps in urban canyon environments. *Robotics and Automation, IEEE Transactions on*, 19(1):15 – 25, feb 2003.

[27] D. Robson, T. Thom, and P. Godwin. *Radio Navigation and Instrument Flying*. Air Pilot's Manual Series. Air Pilot, 2003.

[28] XSens Technologies. *MVN MotionGrid leaflet http://www.xsens.com/images/stories/products/PDF_Brochures/mvn[Accessed: March 2013]*, 2012.

[29] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *Robotics Automation Magazine, IEEE*, 13(2):99 –110, june 2006.

[30] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *Robotics & Automation Magazine, IEEE*, 13(3):108–117, 2006.

[31] Viet Nguyen, Ahad Harati, and Roland Siegwart. A lightweight slam algorithm using orthogonal planes for indoor mobile robotics. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 658–663. IEEE, 2007.

[32] Mirco Alpen, Klaus Frick, and Joachim Horn. On the way to a real-time on-board orthogonal slam for an indoor uav. *Intelligent Robotics and Applications*, pages 1–11, 2011.

[33] Dirk Hahnel, Wolfram Burgard, Dieter Fox, and Sebastian Thrun. An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *Intelligent Robots and Systems, 2003.(IROS*

2003). Proceedings. 2003 IEEE/RSJ International Conference on, volume 1, pages 206–211. IEEE, 2003.

[34] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *International Joint Conference on Artificial Intelligence*, volume 18, pages 1151–1156. LAWRENCE ERLBAUM ASSOCIATES LTD, 2003.

[35] Tim Bailey, Juan Nieto, and Eduardo Nebot. Consistency of the fastslam algorithm. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 424–429. IEEE, 2006.

[36] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *Robotics, IEEE Transactions on*, 23(1):34–46, 2007.

[37] Austin Eliazar and Ronald Parr. Dp-slam: Fast, robust simultaneous localization and mapping without predetermined landmarks. In *International Joint Conference on Artificial Intelligence*, volume 18, pages 1135–1142. LAWRENCE ERLBAUM ASSOCIATES LTD, 2003.

[38] Bruno Steux and Oussama El Hamzaoui. tinyslam: A slam algorithm in less than 200 lines c-language program. In *Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on*, pages 1975–1979. IEEE, 2010.

[39] Paul J Besl and Neil D McKay. A method for registration of 3-d shapes. *IEEE Transactions on pattern analysis and machine intelligence*, 14(2):239–256, 1992.

[40] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *International journal of computer vision*, 13(2):119–152, 1994.

[41] Chen Yang and Gérard Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992.

[42] Hauke Strasdat, JMM Montiel, and Andrew J Davison. Scale drift-aware large scale monocular slam. In *Proceedings of Robotics: Science and Systems (RSS)*, volume 2, page 5, 2010.

[43] Nikolas Engelhard, Felix Endres, Jürgen Hess, Jürgen Sturm, and Wolfram Burgard. Real-time 3d visual slam with a hand-held rgb-d camera. In *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Vasteras, Sweden*, volume 2011, 2011.

[44] Robert Michalson. Iarc advances state of the art in intelligent autonomous aerial robots. *Unmanned Systems*, October, 2011.

[45] http://iarc.angel-strike.com/ Accessed: Feb 2013.

[46] A. Bachrach, R. He, and N. Roy. Autonomous flight in unknown indoor environments. *International Journal of Micro Air Vehicles*, 1(4):217–228, 2009.

[47] A. Bachrach, S. Prentice, R. He, and N. Roy. Range–robust autonomous navigation in gps-denied environments. *Journal of Field Robotics*, 28(5):644–666, 2011.

[48] D M Sobers Jr, G Chowdhary, and E N Johnson. Indoor navigation for unmanned aerial vehicles. Technical report, DTIC Document, 2009.

[49] S I A Shah. Single camera based vision systems for ground and; aerial robots. pages 33–39, 2010.

[50] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I.L. Grixa, F. Ruess, M. Suppa, and D. Burschka. Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue. *Robotics Automation Magazine, IEEE*, 19(3):46 –56, sept. 2012.

[51] Shaojie Shen, Nathan Michael, and Vijay Kumar. Autonomous multi-floor indoor navigation with a computationally constrained mav. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 20–25. IEEE, 2011.

[52] J Roberts, Timothy Stirling, Jean-Christophe Zufferey, and Dario Floreano. Quadrotor using minimal sensing for autonomous indoor flight. In *Proceedings of the European Micro Air Vehicle Conference and Flight Competition (EMAV2007)*, 2007.

[53] A Ravikanth, CR Raviteja, NNVS Pavan Kumar, Vamsimohan Ch, Vikram R Shah, Hem Rampal, and Kedar Kulkarni. Development of an autonomous aerial vehicle capable of indoor navigation. 2009.

[54] Thomas Brady, Nona Ebrahimi, Matt Edelman, Daniel Ellis, Thomas Manville, Rohan Mehta, Ryan Moore, Anthony Smith, Alex Valencourt, Brandon Wagoner, et al. Quadrotor unmanned aerial vehicle for the international aerial robotics competition.

[55] Samir Bouabdallah, Pierpaolo Murrieri, and Roland Siegwart. Towards autonomous indoor micro vtol. *Autonomous Robots*, 18(2):171–183, 2005.

[56] Girish Chowdhary, David Michael Sobers, Erwan Salaun, John Ottander, and Eric Johnson. Fully autonomous indoor flight relying on only five very low-cost range sensors. *Journal of Aerospace Computing, Information, and Communication*, 10(1):21–31, 2013.

[57] Daniel Mellinger, Nathan Michael, and Vijay Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research*, 31(5):664–674, 2012.

[58] Alex Kushleyev, Daniel Mellinger, and Vijay Kumar. Towards a swarm of agile micro quadrotors. In *Robotics: Science and Systems (RSS)*, 2012.

[59] Markus Hehn and Raffaello D'Andrea. A flying inverted pendulum. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 763–770. IEEE, 2011.

[60] Robin Ritz, Mark W Müller, Markus Hehn, and Raffaello D'Andrea. Cooperative quadrocopter ball throwing and catching. In *Intelligent Robots and Systems*

(IROS), 2012 IEEE/RSJ International Conference on, pages 4972–4978. IEEE, 2012.

[61] TheDmel (Youtube Channel). http://youtu.be/MvRTALJp8DM [Accessed: Mar 2013].

[62] TheDmel (Youtube Channel). www.youtu.be/YQIMGV5vtd4 [Accessed: Mar 2013].

[63] UAS Vision. Japan's unmanned flying ball - official launch http://www.uasvision.com/2011/10/27/japans-unmanned-flying-ball-official-launch/ [accessed: May 2013]. Technical report, October 2011.

[64] http://apocalypsedaily.com/wp-content/uploads/2011/10/20111028-010304.jpg [Accessed: Apr 2013].

[65] Microdrones GmbH. www.microdrones.com [Accessed: Mar 2013].

[66] Draganfly Innovations Inc. http://www.draganfly.com/uav-helicopter/draganflyer-x6/ [Accessed: Mar 2013].

[67] Michael Warren, David Mckinnon, Hu He, Arren Glover, Michael Shiel, and Ben Upcroft. Large scale monocular vision-only mapping from a fixed-wing suas. 2012.

[68] Stefan Winkvist, Emma Rushforth, and Ken Young. Towards an autonomous indoor aerial inspection vehicle. *Industrial Robot: An International Journal*, 40(3):196–207, 2013.

[69] D. Wired UK Hambling. http://www.wired.co.uk/magazine/archive/2013/03/start/daring-drone 19th March 2013 [Accessed May 2013].

[70] Stefan Winkvist, Tim Fletcher, Reuben Williams, Matthew Rooke, James Williams, Oliver Vogel, Alex Bunn, and Julian Faulkner. Warwick mobile robotics, search & rescue project 2008-2009 - systems design report http://www.mobilerobotics.warwick.ac.uk/projects/rescue/reports/0809reports/. April 2009.

[71] J. Park and S. Mackay. *Practical Data Acquisition for Instrumentation and Control Systems*. Elsevier Science, 2003.

[72] Feng Lu and Evangelos Milios. Robot pose estimation in unknown environments by matching 2d range scans. *Journal of intelligent & robotic systems*, 18(3):249–275, 1997.

[73] K.S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-9(5):698–700, 1987.

[74] A. Nüchter. *3D Robotic Mapping: The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom*. Springer Tracts in Advanced Robotics. Springer, 2009.

[75] Donald JR Meagher. *Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer*. Electrical and Systems Engineering Department Rensseiaer Polytechnic Institute Image Processing Laboratory, 1980.

[76] H Medellin, J Corney, JBC Davies, T Lim, and JM Ritchie. An automated system for the assembly of octree models. *Assembly Automation*, 24(3):297–312, 2004.

[77] M S LaValle. http://planning.cs.uiuc.edu/node234.html Author: LaValle, M S [Accessed 5th June 2013].

[78] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications (VISSAPP'09)*, pages 331–340, 2009.

[79] Andress Nüchter, Hartmut Surmann, Kai Lingemann, Joachim Hertzberg, and Sebastian Thrun. 6d slam with an application in autonomous mine mapping. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 2, pages 1998–2003. IEEE, 2004.

[80] Andreas Nüchter, Kai Lingemann, Joachim Hertzberg, and Hartmut Surmann. 6d slam-3d mapping outdoor environments. *Journal of Field Robotics*, 24(8-9):699–722, 2007.

[81] G.J.M. Janssen, P.A. Stigter, and R. Prasad. Wideband indoor channel measurements and ber analysis of frequency selective multipath channels at 2.4, 4.75, and 11.5 ghz. *Communications, IEEE Transactions on*, 44(10):1272 – 1288, oct 1996.

[82] OFCOM. *IR2030 - UK Interface Requirements - Licence Exempt Short Range Devices*, ir2030/27/3 edition, December 2011.

[83] Ira S Faberman. Why 'diversity' isn't enough. http://www.iftrontech.com/information.php?info_id=6 Accessed: Nov 2012.

[84] A. Kajiwara. Line-of-sight indoor radio communication using circular polarized waves. *Vehicular Technology, IEEE Transactions on*, 44(3):487 –493, aug 1995.

[85] Wolfram|Alpha. http://www.wolframalpha.com [Accessed May 2013].

Appendix A

Approximating the Size of the UAV Blimp

The following calculations were done to estimate size the approximate size of the UAV if it were built as a lighter-than-air craft.

These were the density constants used, based on *Normal Temperature and Pressure* (NTP) 20°c, 101.325kPa [85]:-

$$\rho_{air} = 1.204kg/m^3 \tag{A.1}$$

$$\rho_{helium} = 0.1663kg/m^3 \tag{A.2}$$

The UAV's mass was estimated using the table of sensors listed in 4.2 on page 67 with additional 100g mass to account for wiring and fixturing. The platform mass is an estimate of the weight of the battery and motors (excluding envelope):-

$$m_{sensors} = 0.9kg \tag{A.3}$$

$$m_{platform} = 0.5kg \tag{A.4}$$

$$m_{UAV} = m_{sensors} + m_{platform} = 1.4kg \tag{A.5}$$

Therefore the size of the balloon/blimp can be estimated where $V_{envelope}$ is the displacement volume needed.

$$F_{buoyancy} = V_{envelope} * (\rho_{air} - \rho_{helium}) * g \tag{A.6}$$

$$V_{envelope} = \frac{F_{buoyancy}}{(\rho_{air} - \rho_{helium}) * g} \tag{A.7}$$

$$0 = F_{buoyancy} - (m_{UAV} * g) \tag{A.8}$$

$$V_{envelope} = \frac{(m_{UAV} * g)}{(\rho_{air} - \rho_{helium}) * g} = \frac{m_{UAV}}{\rho_{air} - \rho_{helium}} \tag{A.9}$$

$$V_{envelope} = \frac{1.4kg}{1.04kg/m^3} = 1.35m^3 \tag{A.10}$$

If the volume $1.35m^3$ is modelled as a sphere, it would have a diameter of *1.37m*. If modelled as an ellipsoid with a *1m diameter* it would become *2.58m long* shown in fig. A.1.

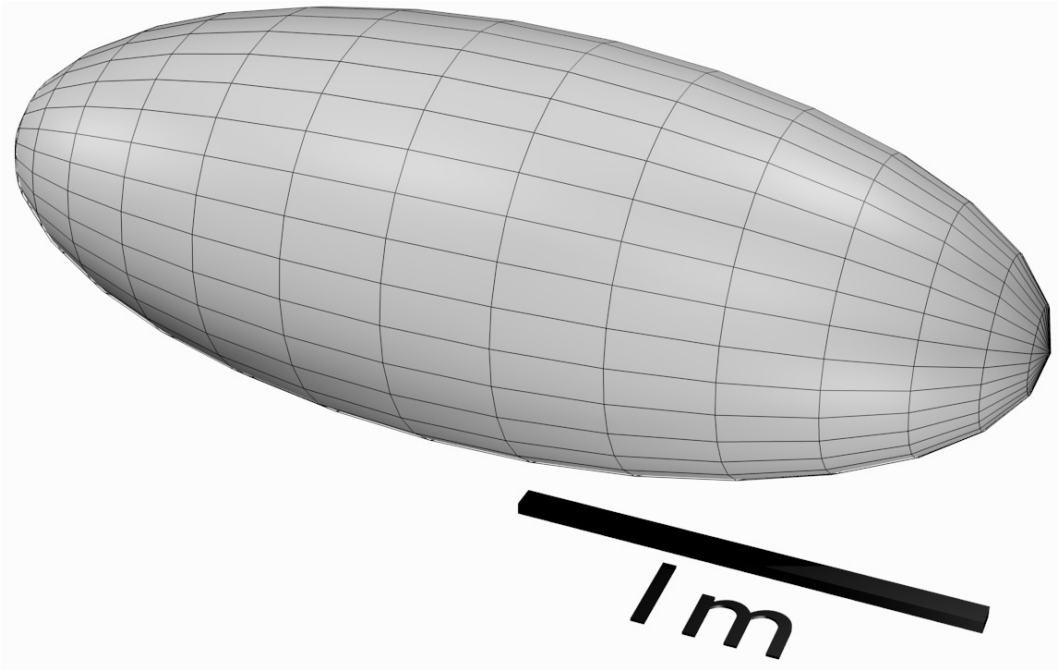


Figure A.1: Visual representation of an ellipsoid representing the size needed for the UAV with the semi-axis of 1.29m, 0.5m, 0.5m

Appendix B

The Developed SLAM Algorithm

The class *AlignmentThread.java* handles the SLAM process and calls the ICP based iterative functions through *ICPFlann.java* on page 197. The custom code which interfaces FLANN with the JAVA code is listed on page 203.

B.1 AlignmentThread.java

```
1  package HeliServer.ScanMatching;
2
3  import HeliServer.Blackbox.BlackboxItem;
4  import HeliServer.Messenger.Messenger;
5  import HeliServer.Movement.RobotPose;
6  import java.io.BufferedWriter;
7  import java.io.FileWriter;
8  import java.io.IOException;
9  import java.util.ArrayList;
10
11 public class AlignmentThread implements Runnable {
12
13     private static Messenger messenger = new Messenger("ScanMatching", "
       Alignment");
```

```
14      private static BlackboxItem blackbox = new BlackboxItem("scanmatch", "X,Y"
        );
15      private static LidarData anchor;
16      private static LidarData scan;
17      private static LidarData secondscan;
18      private static float []  initialposition  ;
19      private static double initialheading  = 0;
20      private static ICPFlann icp = new ICPFlann();
21      public static boolean enabled = true;
22      public static Thread at;
23      private static boolean hasworld = false;
24      float []  lastscanposition  = new float[3];
25      double lastscanheight  = 0;
26      int counter = 0;
27
28      public AlignmentThread() {
29      }
30
31      public void run() {
32          while (enabled) {
33              try {
34                  try {
35                      Thread.sleep(20);
36                  } catch (InterruptedException ex) {
37                  }
38                  if (!hasworld) {
39                      continue;
40                  }
41                  boolean donotrecord = false;
42
```

```
43                  setScan(RobotPose.lidar_data);
44                  icp.setscan(scan.getLidarXYZ());
45                  icp.sett(icp.getabst());
46                   initialposition  = icp.getabst().clone();
47                  icp.applytransform();
48
49                  double errorbefore = 0.000001;
50
51                  for (int iterate = 0; iterate < 14; iterate++) {
52                      icp.iterate();
53                      if (iterate == 0) {
54                          errorbefore = icp.calcerrorFunction();
55                      }
56
57                      icp.applytransform();
58
59                      if (distance(icp.getabst(), initialposition) > 1) {
60                          icp.rollbackabsT( initialposition );
61                          donotrecord = true;
62                          System.out.println("Jumping scan due to distance moved."
                            );
63                          break;
64                      }
65                      icp.acceptmove();
66                  }
67
68                  double errorafter = icp.calcerrorFunction();
69                  double error = errorafter / errorbefore;
70
71                  if (error < 0.0 || error > 1.05 || !hasworld) {
```

```
72                  donotrecord = true;
73                  System.out.println("Jumping scan due to low correlation " +
                    error);
74              }
75
76              if (donotrecord) {
77                  continue;
78              }
79
80              scan.scanmatched = true;
81              scan.movement_x = icp.getabst()[0];
82              scan.movement_y = icp.getabst()[1];
83
84              icp.applytransform();
85              scan.updateLidarXYZ(icp.getscan());
86
87              double dist = distance(icp.getabst(), lastscanposition);
88
89              if (dist > 0.5 || Math.abs(scan.movement_z - lastscanheight) >
                0.20) {
90                  lastscanposition = icp.getabst().clone();
91                  lastscanheight = scan.movement_z;
92                  anchor.addXYZpoints(icp.getscan());
93                  icp.setworld(anchor.getLidarXYZ());
94              }
95
96              double[] temp = {icp.getabst()[1], icp.getabst()[0]};
97              RobotPose.updateScanMatching(temp);
98              blackbox.log(temp[0] + "," + temp[1]);
99          } catch (Exception e) {
```

```
100                 messenger.printError("Caught error: " + e.getMessage());
101             }
102         }
103     }
104
105     private static double[][] toDouble(float [][] input) {
106         double[][] output = new double[input.length][input[0].length];
107
108         for (int i = 0; i < input.length; i++) {
109             for (int j = 0; j < input[0].length; j++) {
110                 output[i][j] = (double) input[i][j];
111
112             }
113         }
114         return output;
115     }
116
117     private static float [][] toFloat(double[][] input) {
118         float [][] output = new float[input.length][input[0].length];
119
120         for (int i = 0; i < input.length; i++) {
121             for (int j = 0; j < input[0].length; j++) {
122                 output[i][j] = (float) input[i][j];
123
124             }
125         }
126         return output;
127     }
128
```

```
129     private final static float distance(float[] a, float[] b) {//IGNORES Z
        TRAVEL
130         float out = ((a[0] - b[0]) * (a[0] - b[0])) + ((a[1] - b[1]) * (a[1] - b
            [1]));
131         return (float) Math.sqrt(out);
132
133     }
134
135     public static void savedata(String path, float[][] xyzdata) {
136         FileWriter fil ;
137         try {
138             fil = new FileWriter(path);
139
140             BufferedWriter bw = new BufferedWriter(fil);
141
142             for (float[] data : xyzdata) {
143                 for (float d : data) {
144                     bw.write(d + ",");
145                 }
146                 bw.newLine();
147             }
148
149             try {
150                 if (bw != null) {
151                     bw.flush();
152                     bw.close();
153                 }
154             } catch (IOException ex) {
155                 ex.printStackTrace();
156             }
```

```
157         } catch (IOException ex) {
158             System.out. println ("Error  writing  " + ex.getMessage());
159         }
160     }
161
162     public static short [][]  getShortScan() {
163         if (scan == null) {
164             return new short [1][2];
165         }
166         if (scan.getLidarXYZ() == null) {
167             return new short [1][2];
168         }
169
170         ArrayList<short[]> output = new ArrayList<short[]>();
171         for (float [] data : scan.getLidarXYZ()) {
172             short [] temp = {(short) (data[0] * 1000), (short) (data[1] * 1000)};
173             output.add(temp);
174         }
175         return output.toArray(new short[output.size()][output.get(0).length]);
176
177     }
178
179     public static short [][]  getShortWorld() {
180         if (anchor == null) {
181             return new short [1][2];
182         }
183         if (anchor.getLidarXYZ() == null) {
184             return new short [1][2];
185         }
186
```

```
187         ArrayList<short[]> output = new ArrayList<short[]>();
188         for (float [] data : anchor.getLidarXYZ()) {
189             short [] temp = {(short) (data[0] * 1000), (short) (data[1] * 1000)};
190             output.add(temp);
191         }
192         return output.toArray(new short[output.size()][output.get(0).length]);
193
194     }
195
196     public static float [][] getFloatWorld() {
197         return anchor.getLidarXYZ();
198     }
199
200     public static void setAnchor(short [] newanchor) {
201         hasworld = false;
202         anchor = new LidarData();
203         anchor.setSingleLidarScan(newanchor);
204         anchor.calcLidar2DXY();
205         anchor.rotate(-RobotPose.xsens_orientation[1], -RobotPose.
            xsens_orientation[0], -RobotPose.xsens_orientation[2]);
206         messenger.println("Added new anchor");
207         icp.setworld(anchor.getLidarXYZ());
208         icp.resetabsRt();
209         hasworld = true;
210
211     }
212
213     public static void resetWorldMapandRetainPosition() {
214         hasworld = false;
215         anchor = new LidarData();
```

```
216             scan.movement_z = scan.ultrasound_height;
217             scan.ultrasound_height = RobotPose.height_absolute;
218             anchor.setSingleLidarScan(RobotPose.lidar_data);
219             anchor.calcLidar2DXY();
220             anchor.rotate(-RobotPose.xsens_orientation[1], -RobotPose.
                xsens_orientation[0], -RobotPose.xsens_orientation[2]);
221             icp.sett( initialposition );
222             messenger.printError("Tried to recover map, reassigned anchor");
223             icp.setworld(icp.applytransform(anchor.getLidarXYZ()));
224             hasworld = true;
225         }
226
227         public static void setScan(short[] newanchor) {
228             scan = new LidarData();
229             scan.ultrasound_height = RobotPose.height_absolute;
230             scan.movement_z = scan.ultrasound_height;
231             scan.setSingleLidarScan(newanchor);
232             scan.calcLidar2DXY();
233             scan.rotate(-RobotPose.xsens_orientation[1], -RobotPose.
                xsens_orientation[0], -RobotPose.xsens_orientation[2]);
234         }
235 }
```

B.2 ICPFlann.java

```
1 package HeliServer.ScanMatching;
2
3 import HeliServer.ScanMatching.octree.Octree;
4 import JNIflann.flann;
5 import java.util.ArrayList;
```

```
6
7  public class ICPFlann {
8
9      private float [][] dataset, scanset, worldset, similarpoints ;
10     private float [] t = new float[3], absolutet = new float[3];
11     private World world;
12     private float errorbefore ;
13     private float errorafter ;
14     private int [] flann_index;
15     private boolean first_iteration = true;
16
17     public void ICPOptimised() {
18     }
19
20     public void iterate () {
21         findSimilarPointsFlann ();
22         t = calcBestTranslation ();
23         t[2] = 0; //Ignore Z travel
24     }
25
26     public void acceptmove() {
27         absolutet [0] += t[0];
28         absolutet [1] += t[1];
29         absolutet [2] += t[2];
30     }
31
32     public void applytransform() {
33         scanset = doTransformation(scanset);
34     }
35
```

```
36      public float [][] applytransform(float [][] data) {
37          return doTransformation(data);
38      }
39
40      public void rollbackabsT(float [] inp) {
41          absolutet = inp;
42      }
43
44      public float [] getabst() {
45          return absolutet;
46      }
47
48      public void sett(float [] inp) {
49          t[0] += inp[0];
50          t[1] += inp[1];
51          t[2] += inp[2];
52      }
53
54      public void resetRt() {
55          t = new float[3];
56      }
57
58      public void resetabsRt() {
59          absolutet = new float[3];
60      }
61
62      public float geterrorBefore() {
63          return errorbefore;
64      }
65
```

```
66      public float geterrorAfter () {
67          return errorafter ;
68      }
69
70      public float [][] getscan() {
71          return scanset;
72      }
73
74      public void setscan(float [][] input) {
75          scanset = input;
76           first_iteration = true;
77      }
78
79      public boolean hasMoved() {
80          if (t[0] + t[1] + t[2] < 0.000001) {
81              return true;
82          }
83          return false;
84      }
85
86      public float [][] getworld() {
87          return worldset;
88      }
89
90      public void setworld(float [][] input) {
91          worldset = input;
92          rebuildWorld(input);
93      }
94
95      private void rebuildWorld(float [][] input) {
```

```
96          System.out. println ("BUILDING TREE");
97          world = new World();
98          world.addPoints(input);
99          System.out. println ("Tree built ... ");
100     }
101
102     public float calcerrorFunction ()
103     {
104         float output = 0.0f;
105         for (int i = 0; i < scanset.length; i++) {
106             output += calc3dhypot(scanset[i][0] - (Octree. allpoints [flann_index[
                i ]][0] + t[0]), scanset[i][1] - (Octree. allpoints [flann_index[i ]][1]
                 + t[1]), scanset[i][2] - (Octree. allpoints [flann_index[i ]][2] + t
                [2]));
107         }
108         output /= scanset.length;
109         return output;
110     }
111
112     private float [] calcBestTranslation () {
113
114         float [] cm = new float[3];
115         float [] cd = new float[3];
116         for (int i = 0; i < scanset.length; i++) {
117             cd[0] += scanset[i][0];
118             cd[1] += scanset[i][1];
119             cd[2] += scanset[i][2];
120
121             cm[0] += Octree.allpoints[flann_index[i ]][0];
122             cm[1] += Octree.allpoints[flann_index[i ]][1];
```

```
123             cm[2] += Octree.allpoints[flann_index[i ]][2];
124         }
125
126         cm[0] /= (float) scanset.length;
127         cm[1] /= (float) scanset.length;
128         cm[2] /= (float) scanset.length;
129
130         cd[0] /= (float) scanset.length;
131         cd[1] /= (float) scanset.length;
132         cd[2] /= (float) scanset.length;
133
134         float [] out = {cm[0] - cd[0], cm[1] - cd[1], cm[2] - cd[2]};
135         return out;
136     }
137
138     private float [][] doTransformation(float [][] input) {
139         float [][] output = new float[input.length ][3];
140         for (int i = 0; i < input.length; i++) {
141             output[i ][0] = input[i ][0] + t[0];
142             output[i ][1] = input[i ][1] + t[1];
143             output[i ][2] = input[i ][2] + t[2];
144         }
145         return output;
146     }
147
148     private void findSimilarPointsFlann () {
149         if ( first_iteration ) {
150             flann_index = flann.MatchwithNewData(scanset);
151         } else {
152             flann_index = flann.MatchwithTranslation(t);
```

```
153             }
154         }
155
156         private float calc3dhypot(float x, float y, float z) {
157             return (x * x + y * y + z * z);
158         }
159
160         public float distanceMoved() {
161             return (float) Math.sqrt(calc3dhypot(t[0], t[1], t[2]));
162         }
163     }
```

B.3 Custom JAVA-FLANN Interface

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include <jni.h>
5  #include <flann/flann.h>
6
7  int nn;
8  struct FLANNParameters p;
9  float speedup;
10 flann_index_t index_id;
11
12 float *testset;
13 int testset_size;
14 float *world;
15 int world_size;
16
```

```
17 int *matched_points;
18 int matched_points_size;
19
20 int cols;
21 int hasscan = 0;
22 float t[3];
23
24 jfloatArray ToFloatArray(float *arr, int size, JNIEnv *env)
25 {
26         jfloatArray result;
27         result = (*env)->NewFloatArray(env, size);
28         if (!result)
29         {
30                 fprintf(stderr, "Could not allocate memory\n");
31                 return NULL;
32         }
33         (*env)->SetFloatArrayRegion(env, result, 0, size, arr);
34         return result;
35 }
36
37 jintArray ToIntArray(int *arr, int size, JNIEnv *env)
38 {
39         jintArray result;
40         result = (*env)->NewIntArray(env, size);
41         if (!result)
42         {
43                 fprintf(stderr, "Could not allocate memory\n");
44                 return NULL;
45         }
46         (*env)->SetIntArrayRegion(env, result, 0, size, arr);
```

```
47          return result;
48  }
49
50  float *From2dArray(jfloatArray arr, int sizex, int sizey, JNIEnv *env)
51  {
52          float *result = malloc(sizex*sizey*sizeof(float));
53          int x,y;
54          for (y=0; y<sizey; ++y)
55          {
56                  for (x=0; x<sizex; ++x)
57                  {
58                          (*env)->GetFloatArrayRegion(env, arr, 0, sizex*sizey,
                            result);
59                  }
60          }
61          return result;
62  }
63
64
65  float *From2dTo1d(JNIEnv *env, jobjectArray arr, jint sizey, jint sizex)
66  {
67          float *result = malloc(sizex*sizey*sizeof(float));
68          int x, y;
69          for (y=0; y<sizey; ++y)
70          {
71                  jfloatArray row = (*env)->GetObjectArrayElement(env, arr, y);
72                  float tmp[sizex];
73                  (*env)->GetFloatArrayRegion(env, row, 0, sizex, tmp);
74                  for (x=0; x<sizex; ++x)
75                  {
```

```
76                          int index = y*sizex+x;
77                          result[index] = tmp[x];
78                  }
79          }
80          return result;
81  }
82
83
84  JNIEXPORT jintArray JNICALL Java_JNIflann_flann_MatchwithTranslation(JNIEnv
    *env, jobject obj, jfloatArray translate, jint sizex)
85  {
86          float t[sizex]; //3d
87          int i;
88          int result[testset_size*nn];
89          float dists[testset_size*nn];
90          (*env)->GetFloatArrayRegion(env, translate, 0, 1, t);
91
92          for(i = 0; i<testset_size*sizex; i++) testset[i]+= t[i%sizex]; //
            Translate the current data
93
94          flann_find_nearest_neighbors_index(index_id, testset, testset_size,
            result, dists, nn, &p);
95          matched_points = result;
96          jintArray out = ToIntArray(result, testset_size*nn, env);
97          return out;
98  }
99
100 JNIEXPORT jintArray JNICALL Java_JNIflann_flann_MatchwithNewData(JNIEnv *
    env, jobject obj, jobjectArray inp, jint sizex, jint sizey)
101 {
```

```
102             if (hasscan > 0) free( testset );
103             hasscan++;
104             int  result [sizex*nn];
105             float  dists [sizex*nn];
106             testset  = From2dTo1d(env, inp, sizex,  sizey );
107             testset_size  = sizex;
108             flann_find_nearest_neighbors_index(index_id,  testset ,  testset_size ,
                result ,  dists ,  nn,  &p);
109             matched_points = result;
110             matched_points_size = testset_size;
111             jintArray  out = ToIntArray( result ,  testset_size *nn, env);
112             return out;
113     }
114
115     JNIEXPORT void JNICALL Java_JNIflann_flann_setWorld(JNIEnv *env, jobject obj,
        jobjectArray inp, jint sizex ,  jint  sizey )
116     {
117             nn = 1;
118             p = DEFAULT_FLANN_PARAMETERS;
119             p.algorithm  = FLANN_INDEX_KDTREE;
120             p.trees  = 1;
121             p.checks = 256;
122             p.log_level  = FLANN_LOG_INFO;
123             cols  = sizey;
124             world = From2dTo1d(env, inp, sizex,  sizey );
125             world_size  = sizex;
126             index_id = flann_build_index(world,  sizex ,  cols ,  &speedup, &p);
127     }
```

Appendix C

Graphs from Flight Testing

C.1 Flight:ID-2

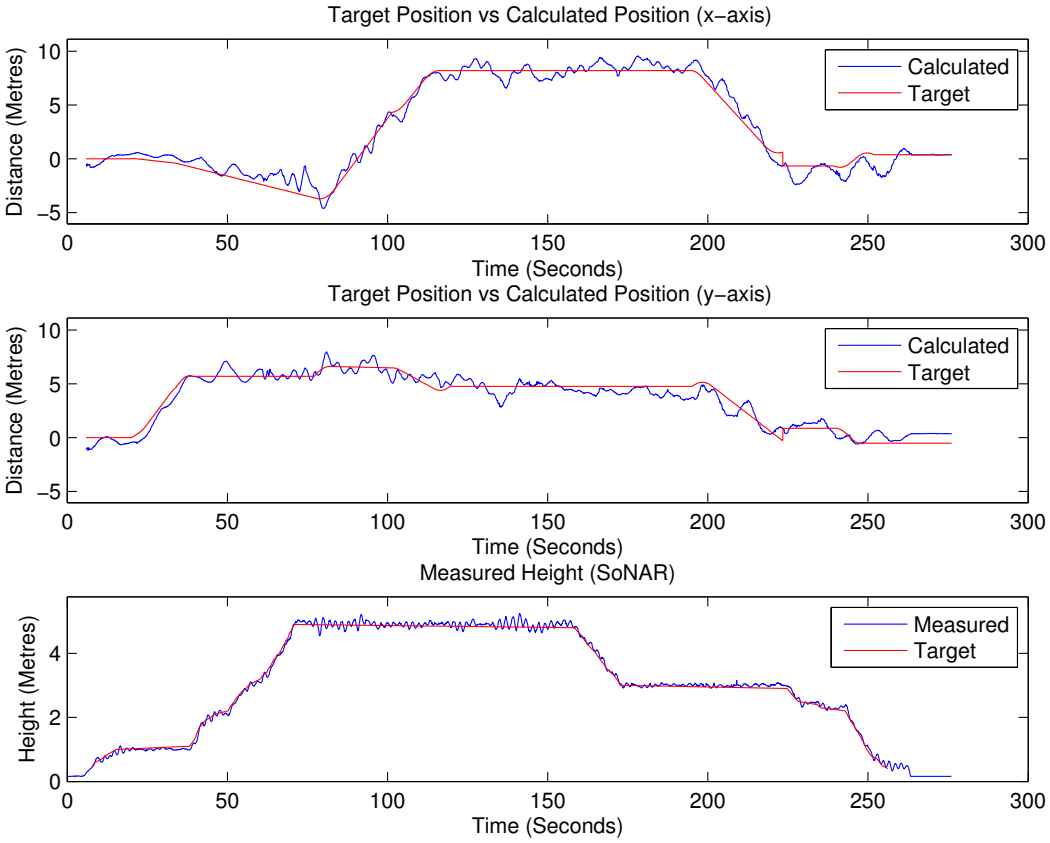
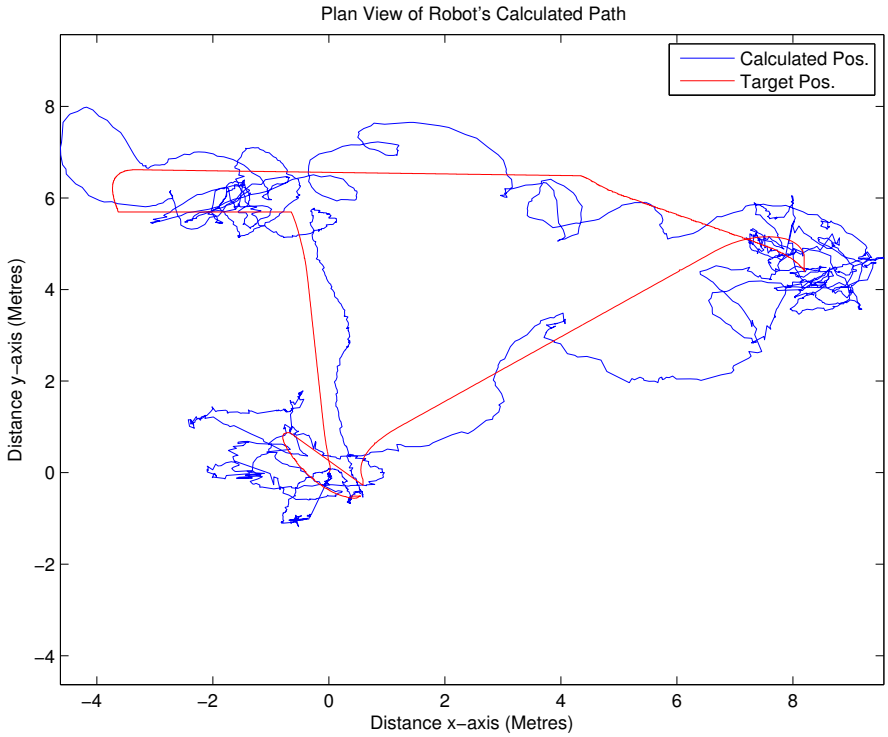
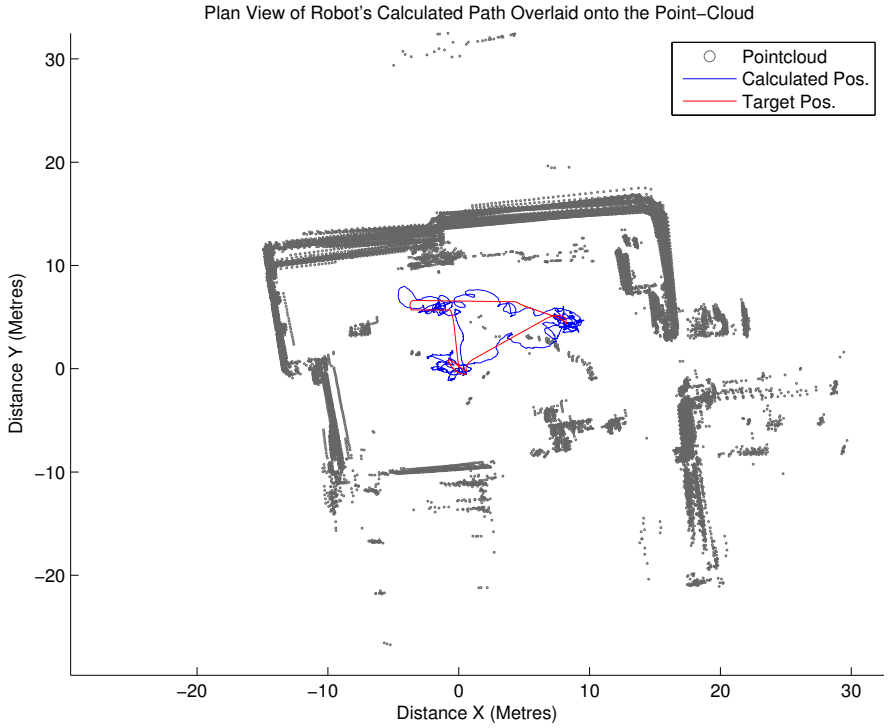


Figure C.1: Flight-ID:Full-2 - UAV's target position and actual position in X,Y and Height axis



(a) Without point-cloud overlay



(b) Overlaid with the point-cloud generated from *Flight-ID:Full-2*

Figure C.2: Flight-ID:Full-2 - A plan view of the robot's path

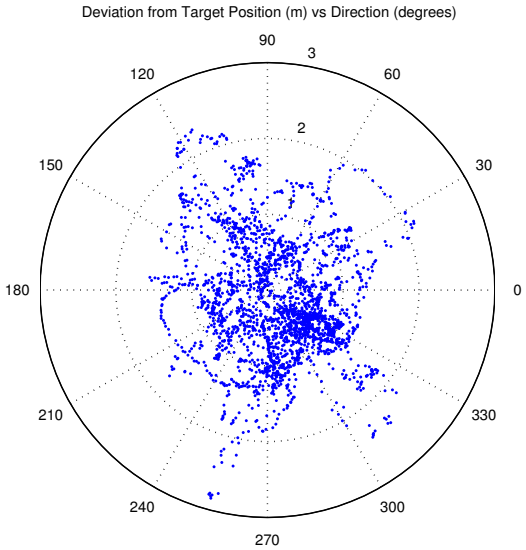


Figure C.3: Flight-ID:Full-2 - UAV's deviation from the target position as a polar plot

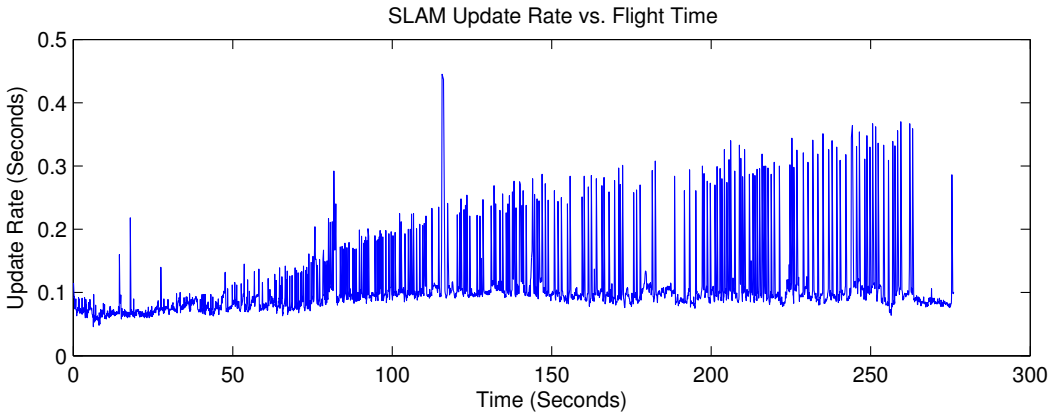


Figure C.4: Flight-ID:Full-2 - Update rate (processing time) of the localisation algorithm during the flight.

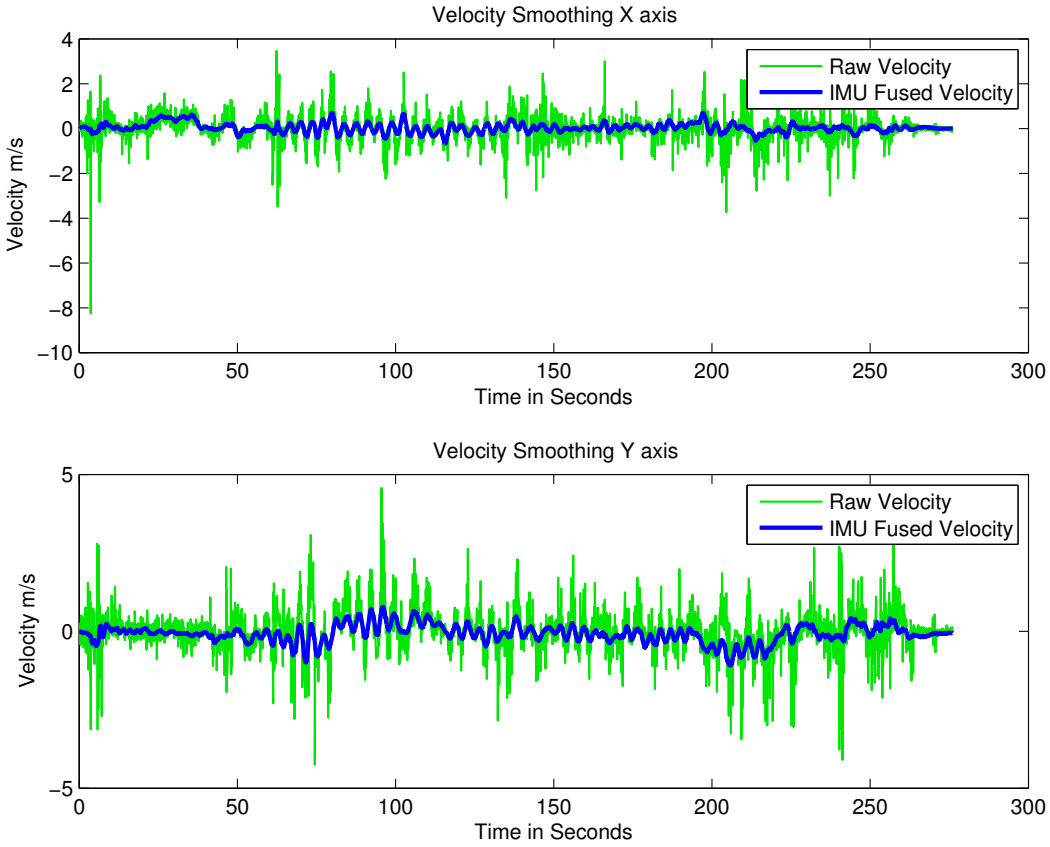


Figure C.5: Flight-ID:Full-2 - Output of the Velocity Estimator throughout the flight

C.2 Flight:ID-3

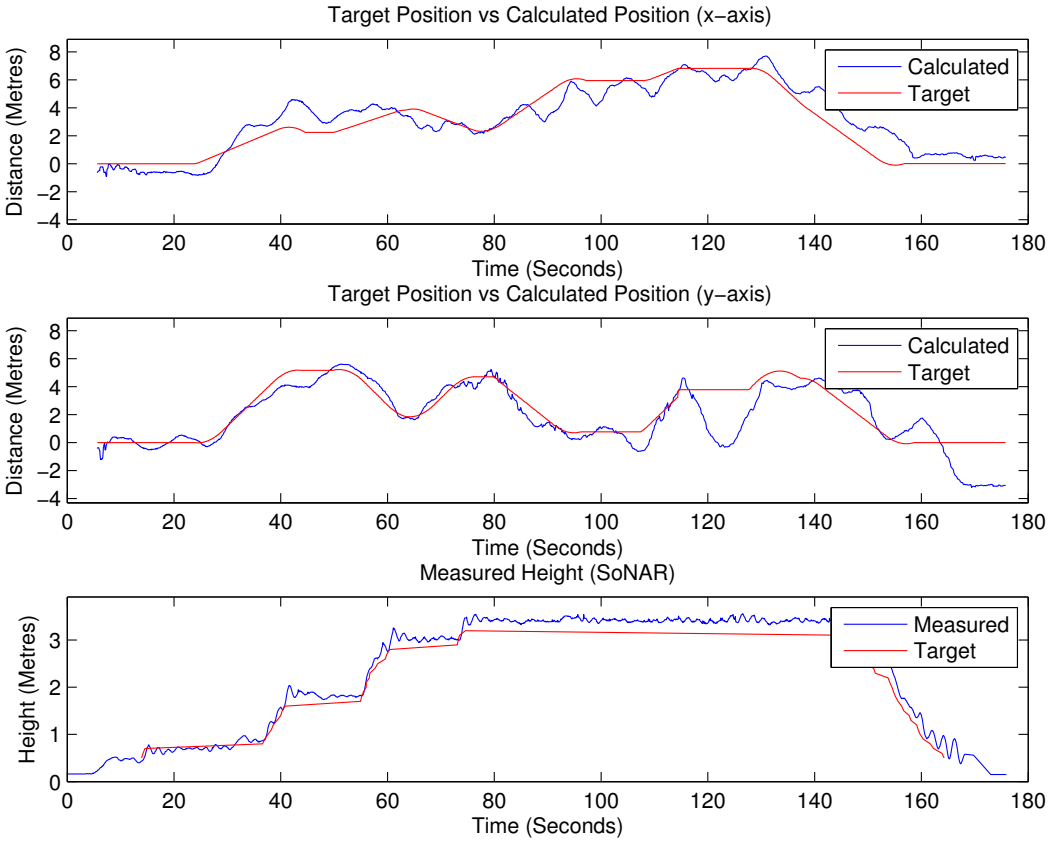
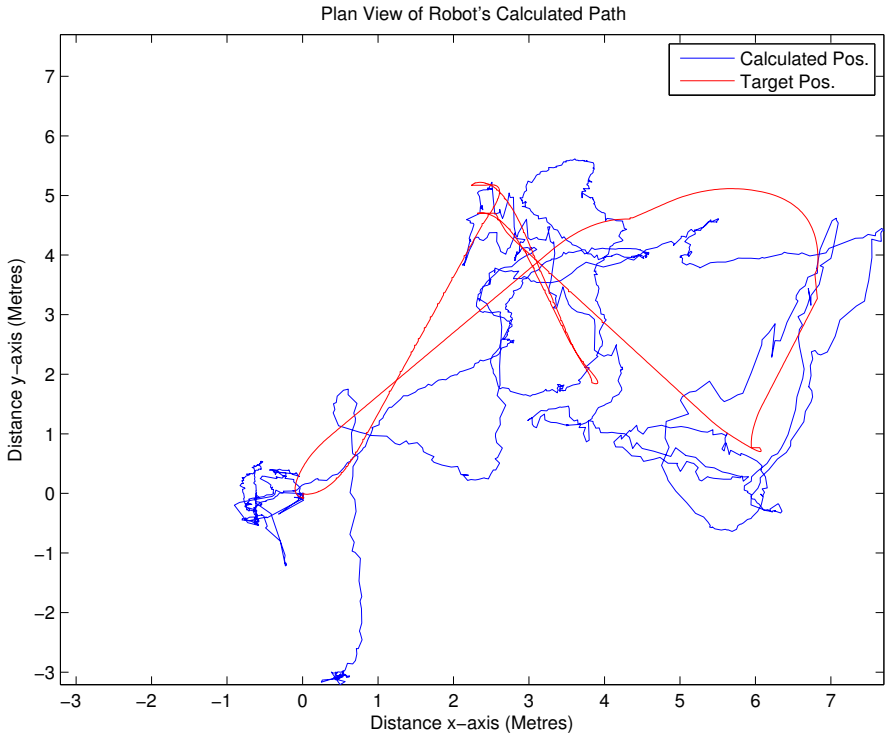
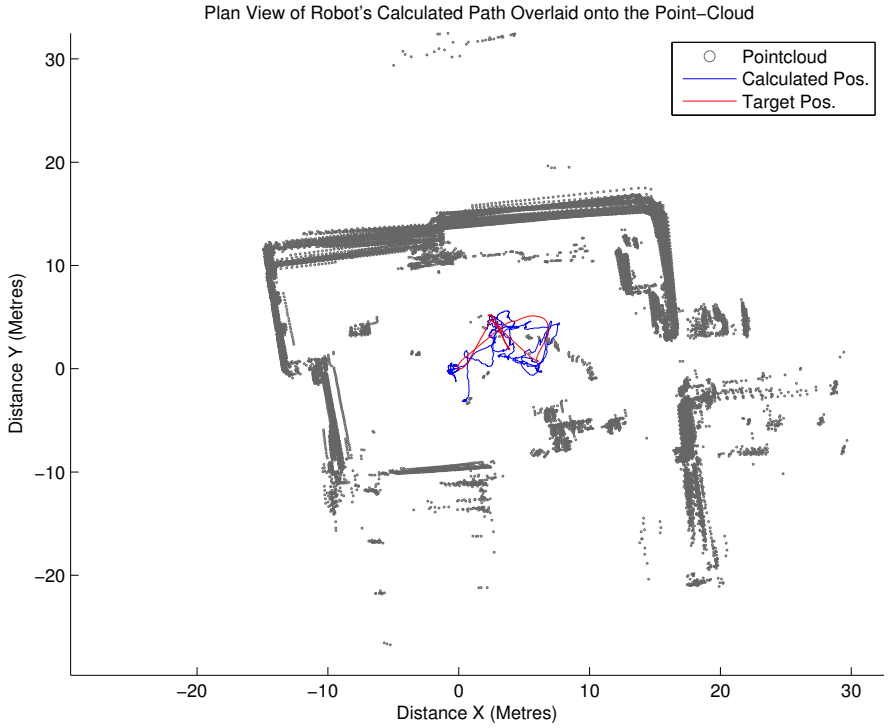


Figure C.6: Flight-ID:Full-3 - UAV's target position and actual position in X,Y and Height axis



(a) Without point-cloud overlay



(b) Overlaid with the point-cloud generated from *Flight-ID:Full-2*

Figure C.7: Flight-ID:Full-3 - A plan view of the robot's path

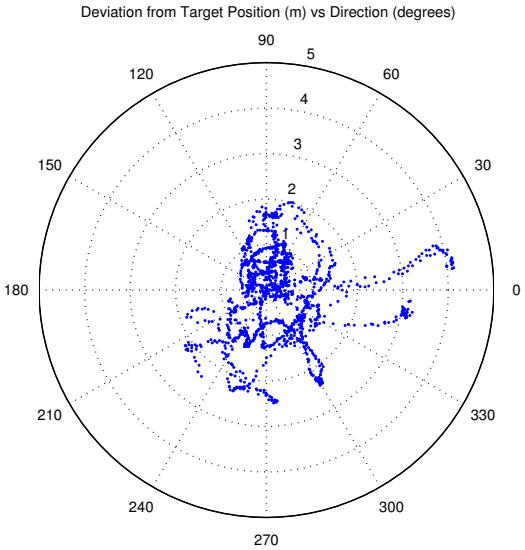


Figure C.8: Flight-ID:Full-3 - UAV's deviation from the target position as a polar plot

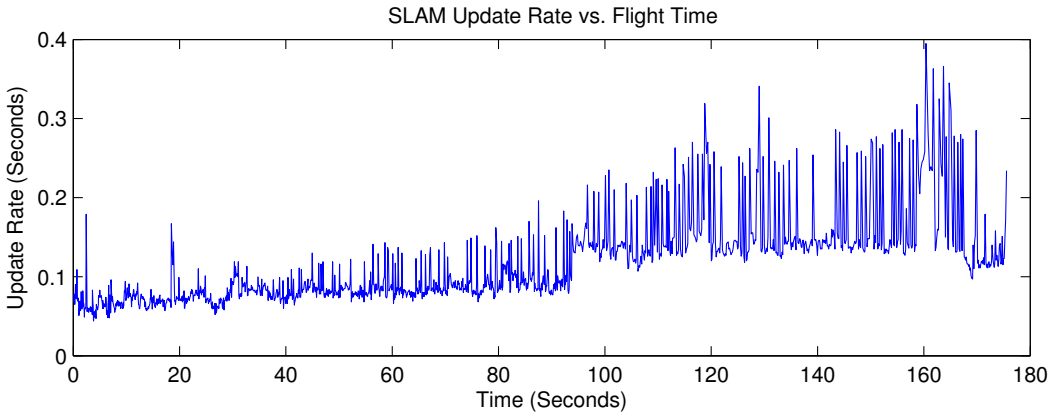


Figure C.9: Flight-ID:Full-3 - Update rate (processing time) of the localisation algorithm during the flight.

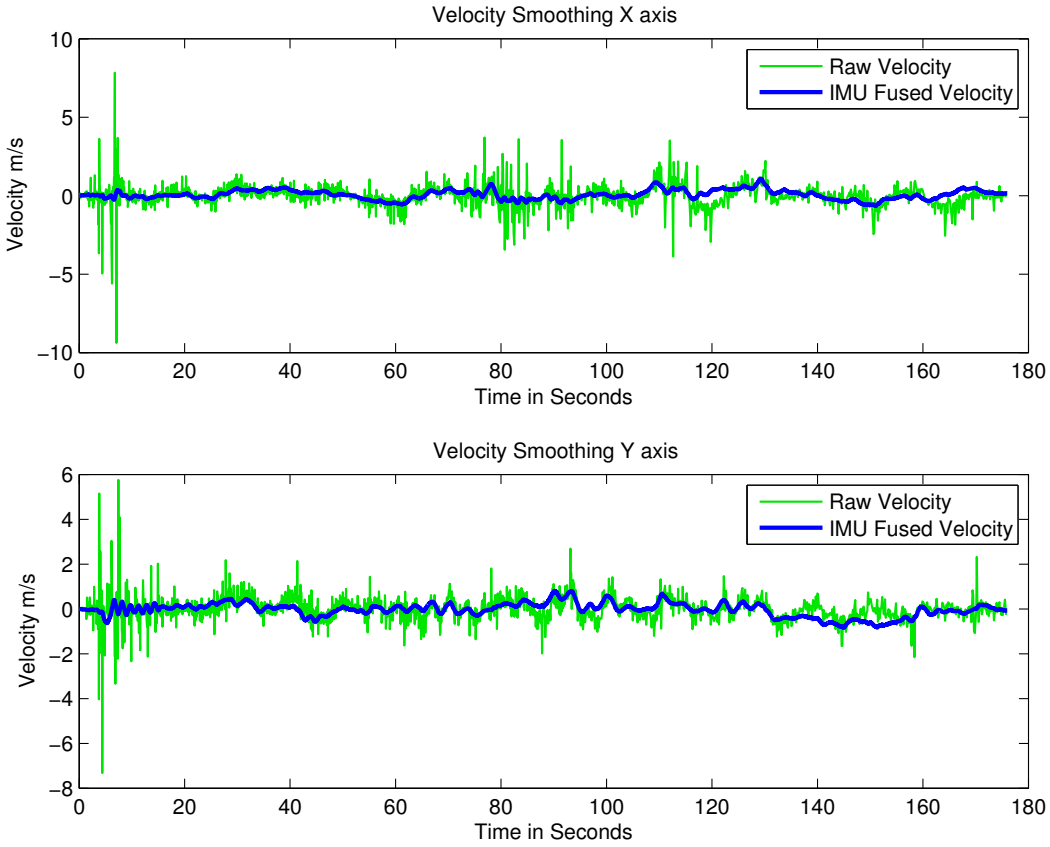


Figure C.10: Flight-ID:Full-3 - Output of the Velocity Estimator throughout the flight

C.3 Flight:ID-4

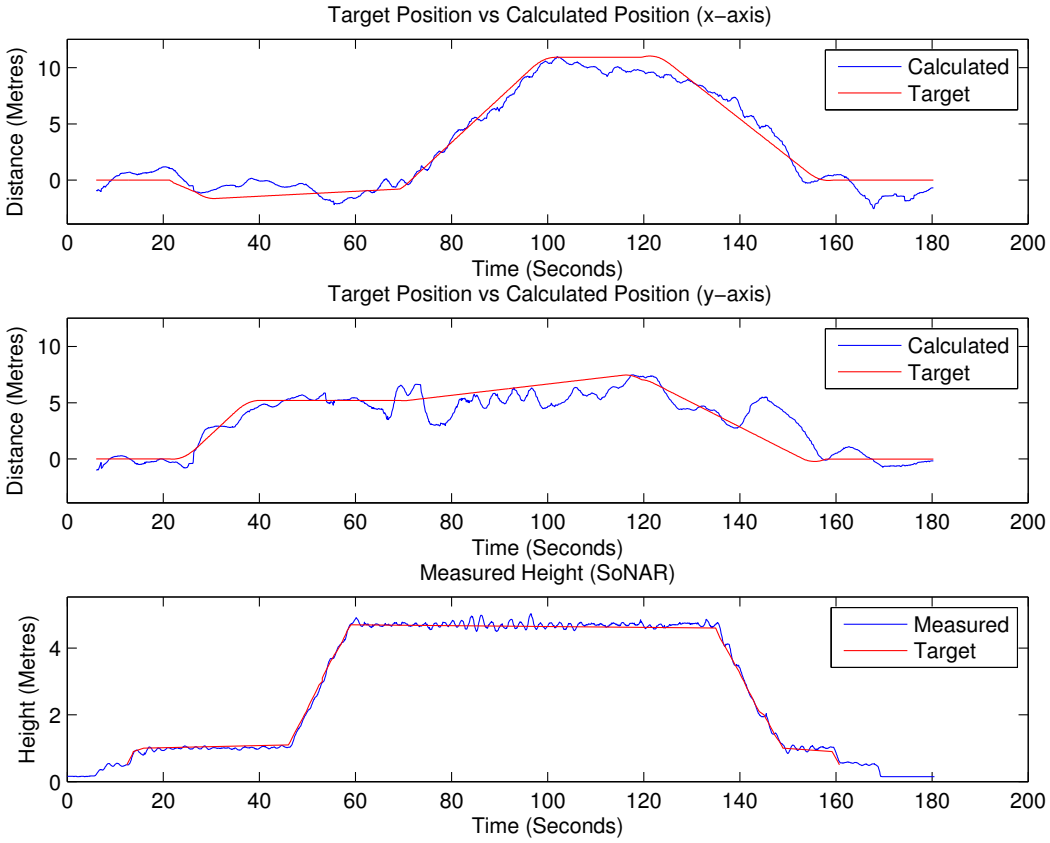
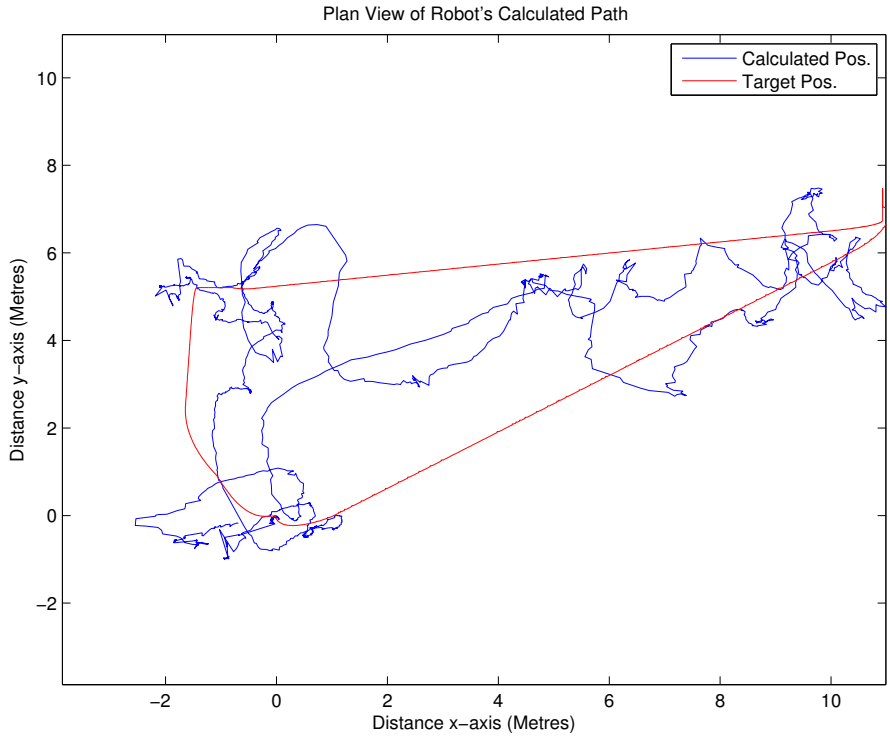
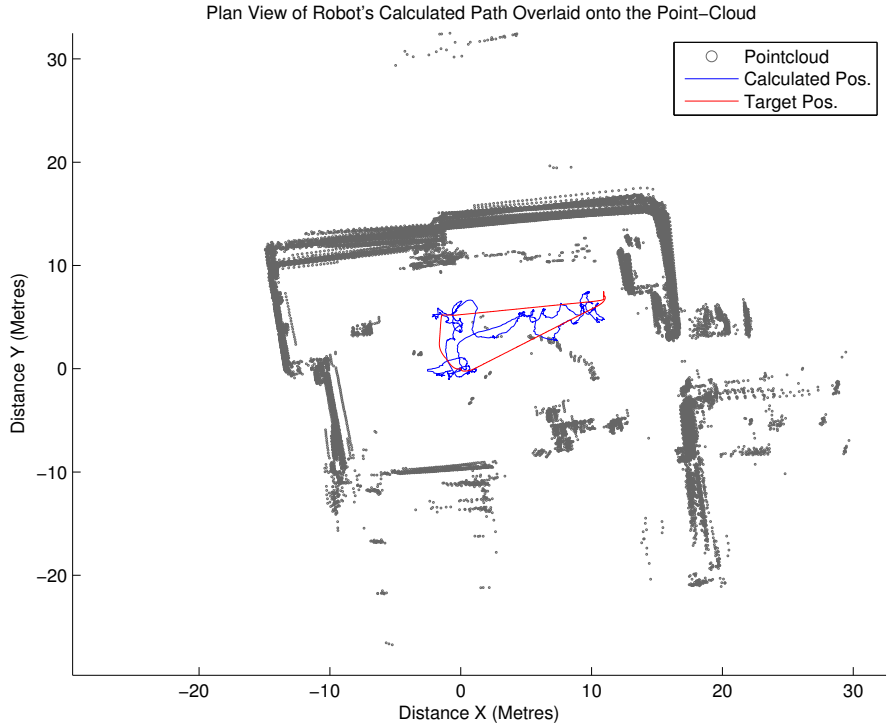


Figure C.11: Flight-ID:Full-4 - UAV's target position and actual position in X,Y and Height axis



(a) Without point-cloud overlay



(b) Overlaid with the point-cloud generated from *Flight-ID:Full-2*

Figure C.12: Flight-ID:Full-4 - A plan view of the robot's path

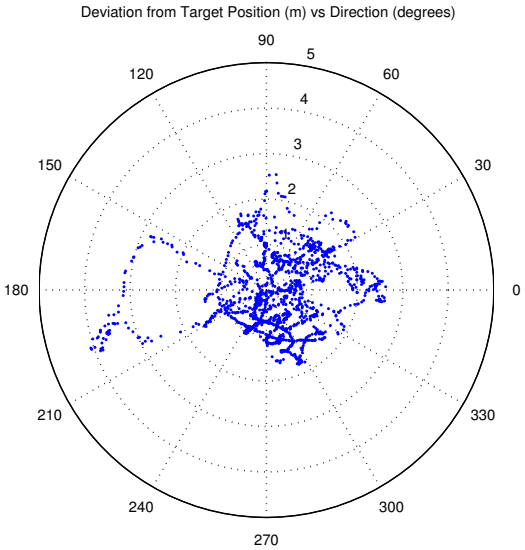


Figure C.13: Flight-ID:Full-4 - UAV's deviation from the target position as a polar plot

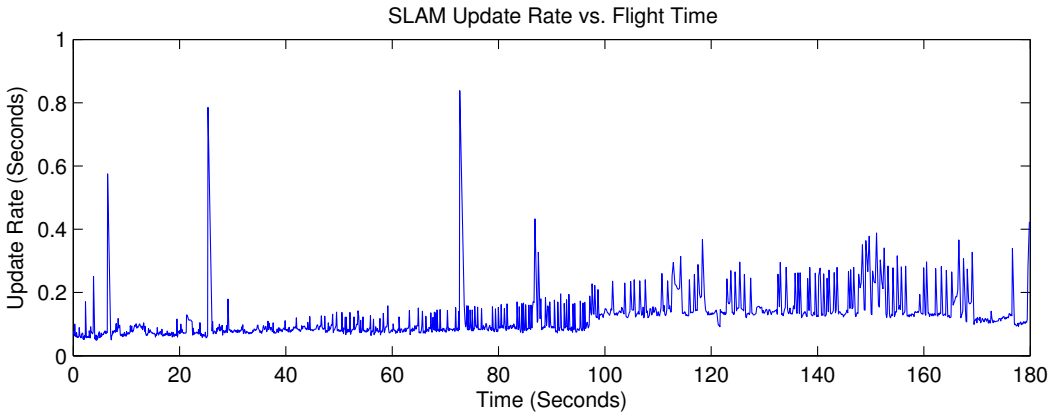


Figure C.14: Flight-ID:Full-4 - Update rate (processing time) of the localisation algorithm during the flight.

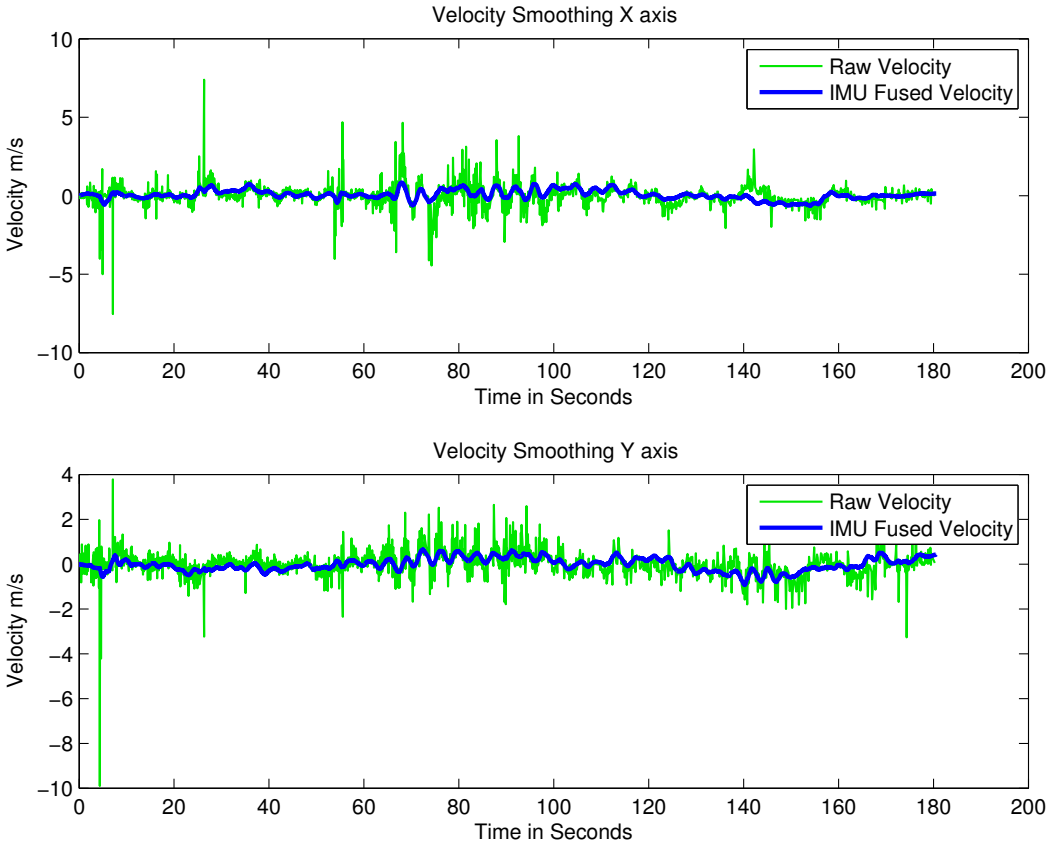


Figure C.15: Flight-ID:Full-4 - Output of the Velocity Estimator throughout the flight

C.4 Flight:ID-5

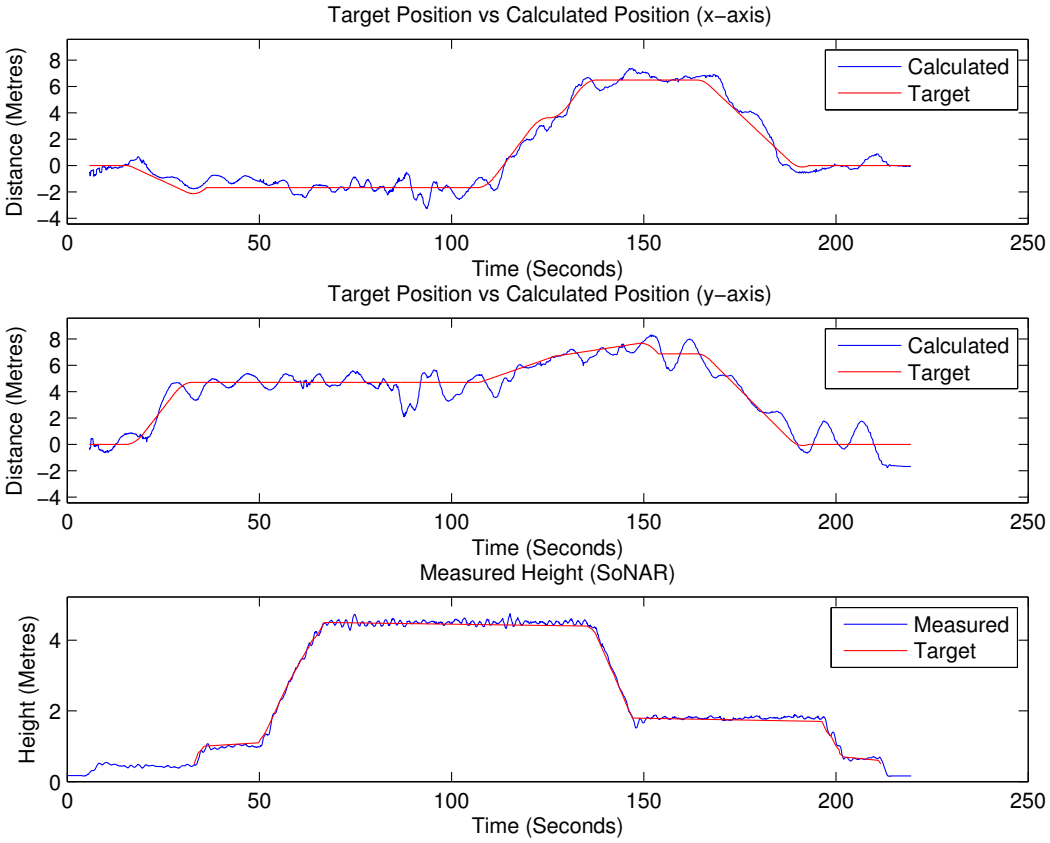
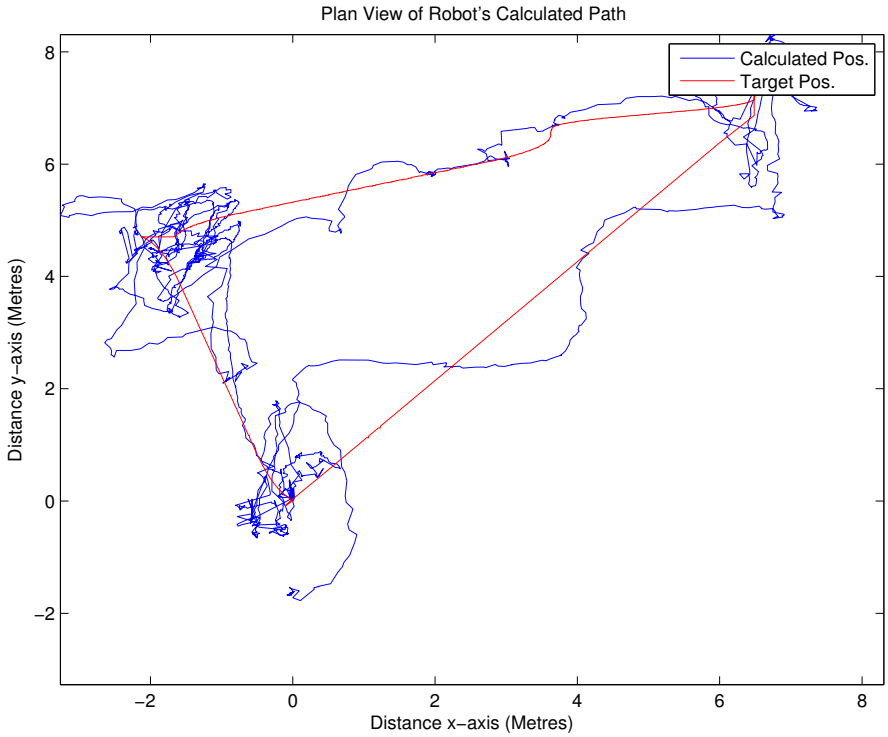
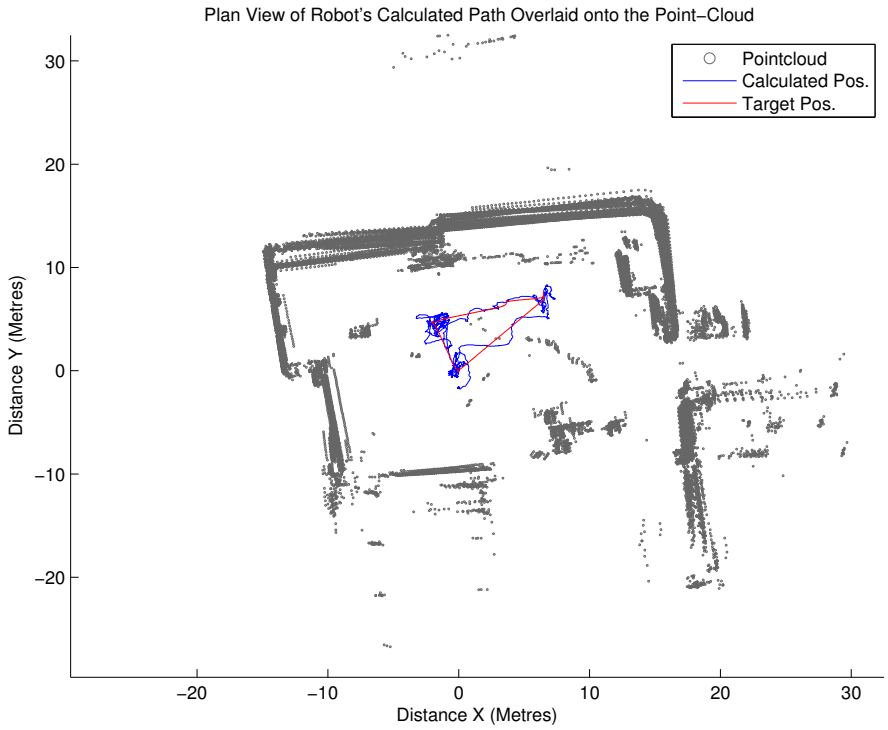


Figure C.16: Flight-ID:Full-5 - UAV's target position and actual position in X,Y and Height axis



(a) Without point-cloud overlay



(b) Overlaid with the point-cloud generated from *Flight-ID:Full-2*

Figure C.17: Flight-ID:Full-5 - A plan view of the robot's path

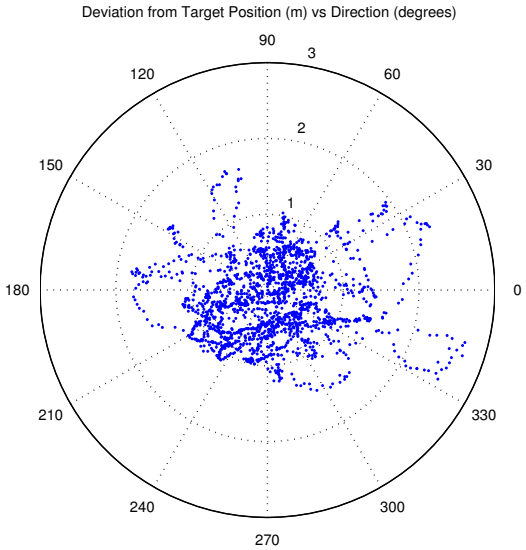


Figure C.18: Flight-ID:Full-5 - UAV's deviation from the target position as a polar plot

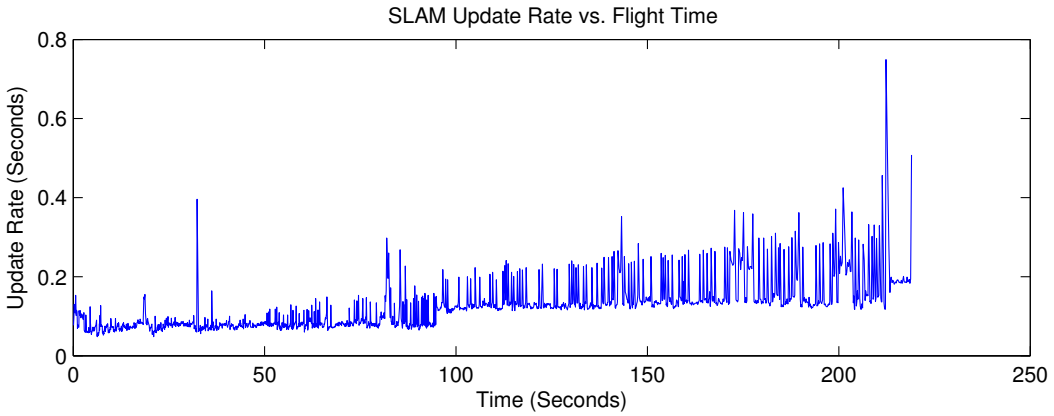


Figure C.19: Flight-ID:Full-5 - Update rate (processing time) of the localisation algorithm during the flight.

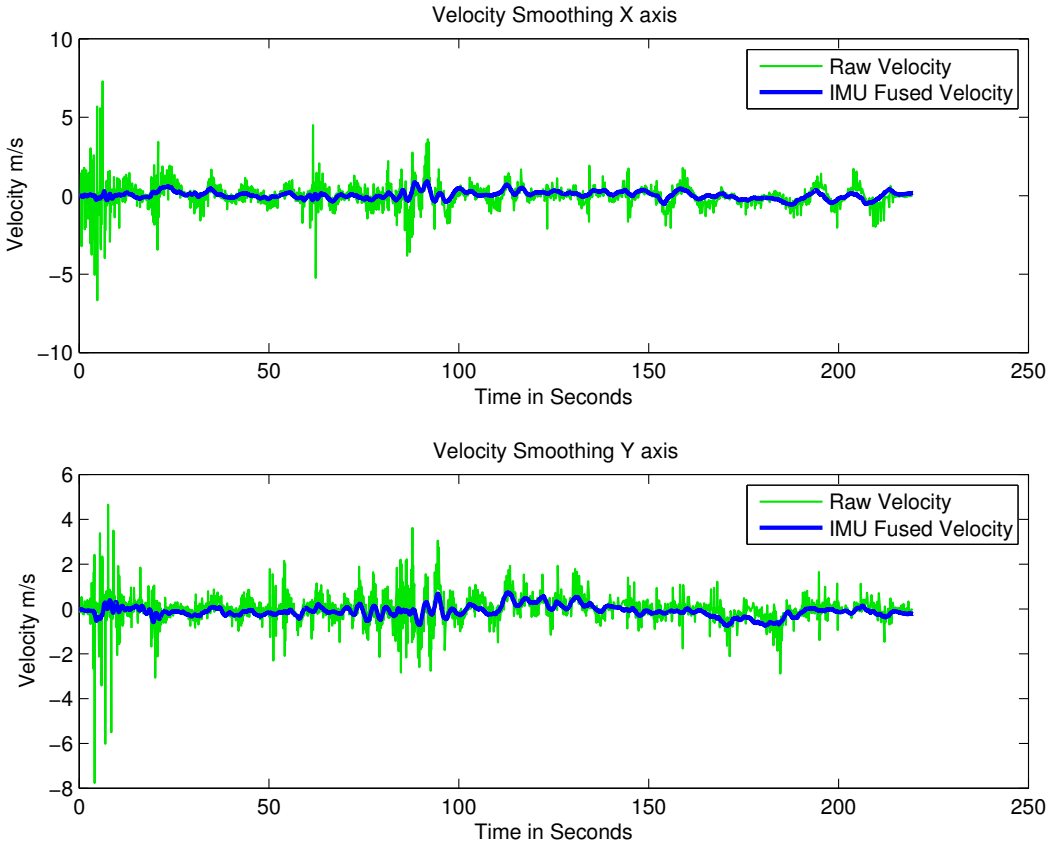


Figure C.20: Flight-ID:Full-5 - Output of the Velocity Estimator throughout the flight

C.5 Flight:ID-6

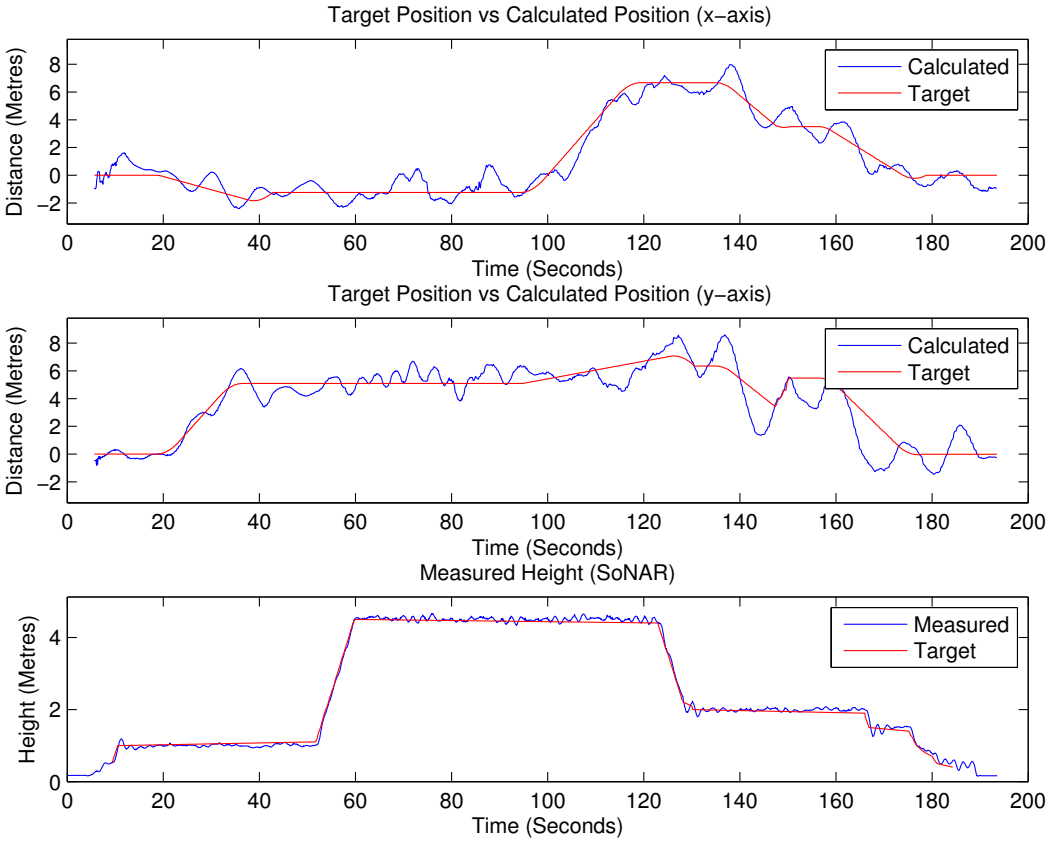
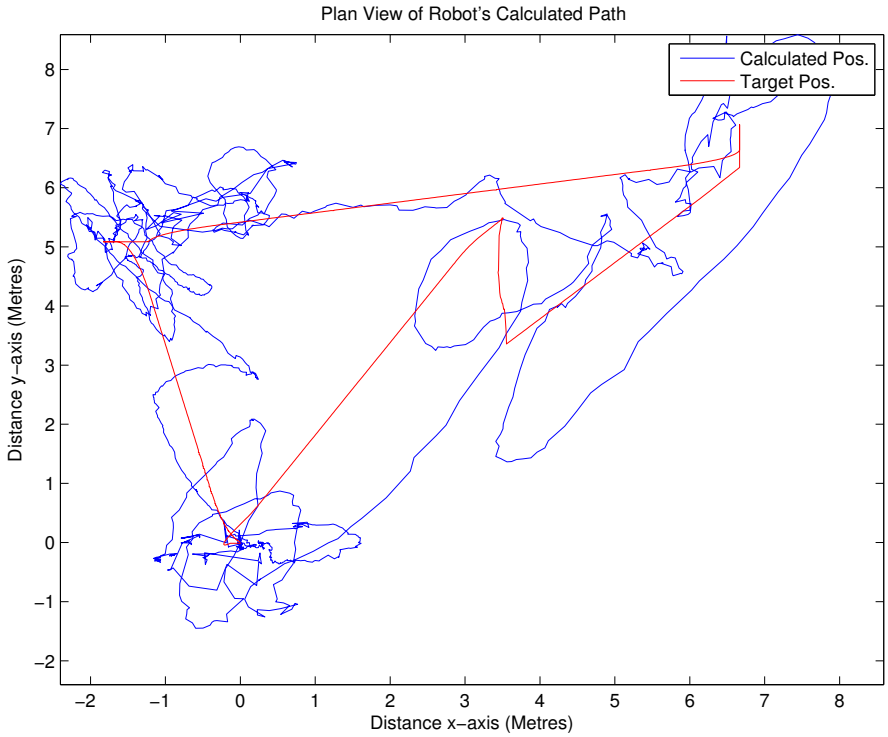
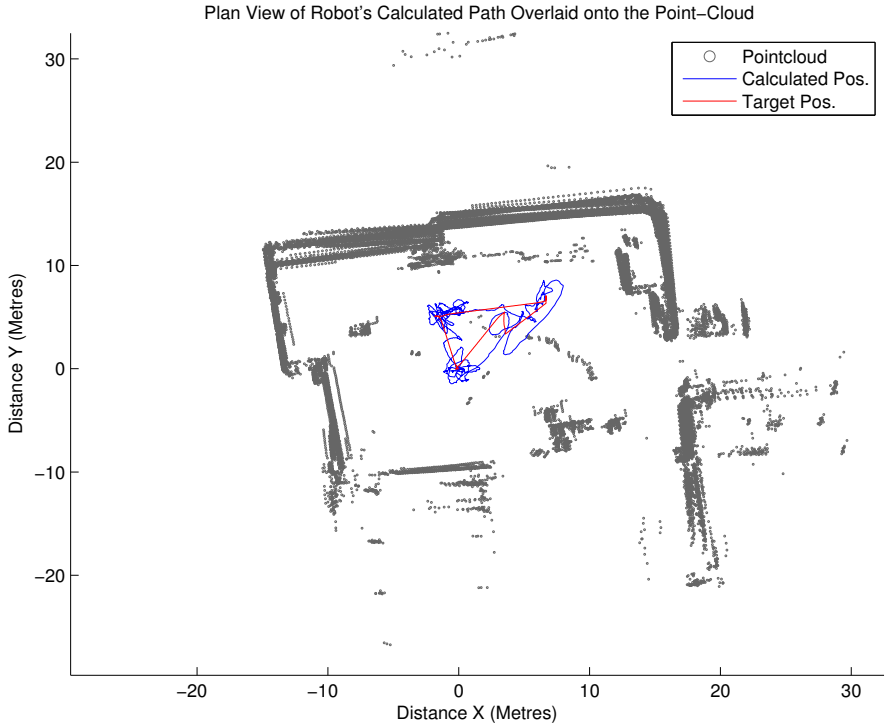


Figure C.21: Flight-ID:Full-6 - UAV's target position and actual position in X,Y and Height axis



(a) Without point-cloud overlay



(b) Overlaid with the point-cloud generated from *Flight-ID:Full-2*

Figure C.22: Flight-ID:Full-6 - A plan view of the robot's path

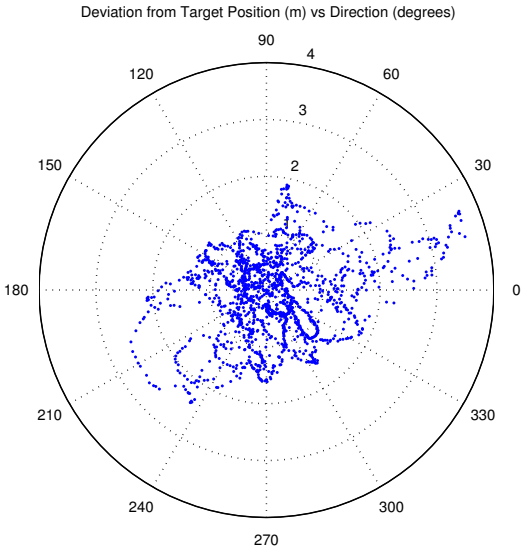


Figure C.23: Flight-ID:Full-6 - UAV's deviation from the target position as a polar plot

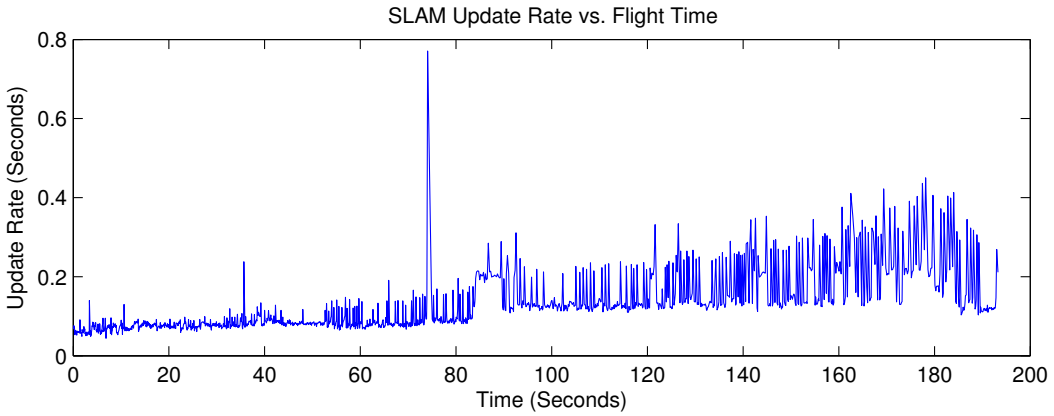


Figure C.24: Flight-ID:Full-6 - Update rate (processing time) of the localisation algorithm during the flight.

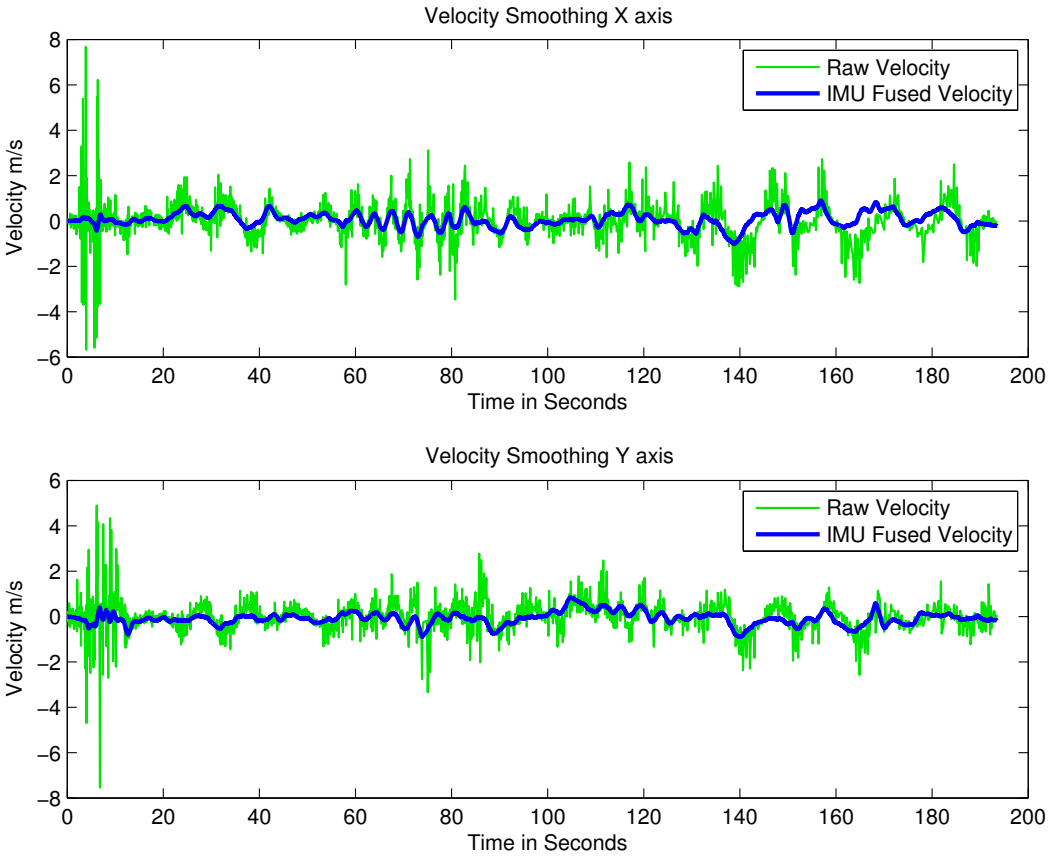


Figure C.25: Flight-ID:Full-6 - Output of the Velocity Estimator throughout the flight

Appendix D

Videos

D.1 Flight Test Videos

Flight	Location of Video
Flight-ID:Full-1 (Crash)	DVD: /Flight-Testing/Full-1.mp4 Web: `http://thesis.steeps.net/full-1/`
Flight-ID:Full-2	DVD: /Flight-Testing/Full-2.mp4 Web: `http://thesis.steeps.net/full-2/`
Flight-ID:Full-3	DVD: /Flight-Testing/Full-3.mp4 Web: `http://thesis.steeps.net/full-3/`
Flight-ID:Full-4	DVD: /Flight-Testing/Full-4.mp4 Web: `http://thesis.steeps.net/full-4/`
Flight-ID:Full-5	Not Available
Flight-ID:Full-6	Not Available
Close Quarter Flying	DVD: /Flight-Testing/Gantry.mp4 Web: `http://thesis.steeps.net/gantry/`

D.2 Testing of Localisation Algorithm

Flight-ID	Location of Video
Visual Demonstration of the SLAM algorithm	DVD: /Localisation/SLAM.mp4 Web: `http://thesis.steeps.net/slam/`
Environmental Transition	DVD: /Localisation/Pit.mp4 Web: `http://thesis.steeps.net/pit/`
Auditorium	DVD: /Localisation/Auditorium.mp4 Web: `http://thesis.steeps.net/auditorium/`